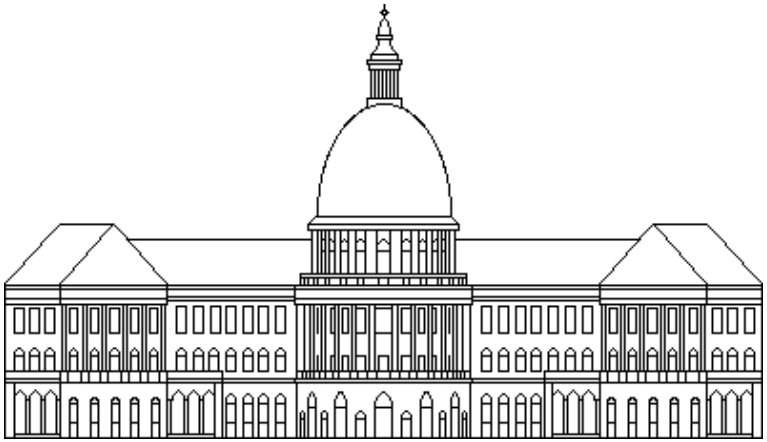
MATLAB an der ETH



Ingenieur-Tool

Software für Numerische Mathematik

Thomas Schubiger
Institut für Mess- und Regeltechnik
ETH Zentrum
CH-8092 Zürich

25.02.1999

Inhaltsverzeichnis

1 MATLAB-Einstieg

Nachdem Sie einige einleitende Worte über Sinn und Zweck diese Kurses gelesen haben, können Sie in diesem Kapitel die ersten Kenntnisse erwerben, um mit MATLAB zu arbeiten. Unter anderem lernen Sie, das sogenannte “Command Window” zu bedienen und die ersten mathematischen Operationen auszuführen. Allmählich machen Sie sich mit der MATLAB-Umgebung vertraut.

1.1 Einführung

1.1.1 Ziel dieses Kurses

Nach Abschluss dieses Kurses sind Sie in der Lage, numerische Berechnungen anhand von MATLAB auszuführen und die erhaltenen Resultate graphisch darzustellen. Durch den graduellen Aufbau der Grundlagen und die zahlreichen konkreten Beispiele haben wir versucht, Ihnen die Fähigkeiten weiterzugeben, selbständig sowohl die wachsende Zahl der auf MATLAB basierenden Software-Bibliotheken (s. Toolbox) zu benutzen, als auch andere Softwarepakete für numerische Berechnungen rasch und effizient für Ihre Bedürfnisse einzusetzen.

Wir wünschen Ihnen viel Spass beim Lernen,
Die Mitglieder des Instituts für Mess- und Regeltechnik

25.02.1999, Thomas Schubiger

1.1.2 Was ist MATLAB?

MATLAB (aus dem englischen MATrix LABoratory) ist ein Software-Paket für numerische Berechnungen. Es zeichnet sich durch die ausgesprochene Effizienz in der Durchführung von Operationen der Linearen Algebra, welche wiederum die Grundlage für die meisten numerischen Berechnungsmethoden sind. Weitere Merkmale sind

- die offene Struktur: praktisch jede enthaltene Funktion, ausgenommen die Grundlagenoperationen der Linearen Algebra, kann vom Benutzer modifiziert werden;
- die thematische Gruppierung von Funktionen (Toolboxes), die je nach Bedürfnis dazugekauft werden können;
- die direkte Kopplung an eine graphische Programmierungsoberfläche für Simulationen (Simulink);
- die Möglichkeit der Einbettung von C- oder Fortran-programmierten Routinen.

1.1.3 Wo kommt MATLAB zur Anwendung?

Während die ursprünglichen Anwendungsgebiete von MATLAB die Verarbeitung, Analyse und Visualisierung von aus wissenschaftlichen Experimenten und Simulationen hervorgegangenen Daten waren, stammen mittlerweile die Anwender von MATLAB aus den verschiedensten Fachrichtungen: Natürlich aus den Ingenieurwissenschaften wie der Elektrotechnik, Regeltechnik, Robotik und Verfahrenstechnik, aber auch aus den Naturwissenschaften wie Geologie, Astronomie, Biologie, Chemie oder Physik. Nicht zuletzt sind die zahlreichen Anwendungen aus der Wirtschaft (Bankwesen, Produktionsplanung, Logistik, u.v.m.) zu erwähnen, welche immer mehr an Bedeutung gewinnen.

1.1.4 Was ist eine Toolbox?

Mit MATLAB fällt der grösste Teil der aufwendigen Programmierung von numerischen Prozeduren und Funktionen weg, da eine grosse Zahl von Befehlen und Funktionen bereits vorhanden ist. MATLAB wird mit einer Grundausstattung von Funktionen geliefert, welche die Durchführung sämtlicher Operationen der Linearen Algebra und die Darstellung der Resultate in graphischer Form erlauben. Darüber hinaus wurde im Laufe der Jahre eine sehr umfangreiche Sammlung von nach Anwendungsbereich gruppierten Funktionen kreiert. Eine Toolbox entspricht einer solchen speziellen Gruppe von sogenannten M-Files, explizit zusammengesetzt zur Lösung einer besonderen Klasse von Problemen. Zur Zeit (Sommer 1998) existieren 23 Toolboxes:

- Communications
- Control System
- Financial
- Frequency Domain System Identification
- Fuzzy Logic
- Higher-Order Spectral Analysis
- Image Processing
- LMI Control
- Mapping
- Model Predictive Control
- μ-Analysis and Synthesis
- NAG Foundation
- Neural Network
- Optimization
- Partial Differential Equation
- QFT Control Design
- Robust Control
- Signal Processing
- Spline
- Statistics
- Symbolic/Extended Symbolic Math
- System Identification

1.1.5 Wie ist MATLAB aufgebaut?

MATLAB wurde ursprünglich geschrieben, um den raschen Zugang zu Matrix-Software zu erlauben, welche im Umfeld der LINPACK- und EISPACK-Projekte entwickelt wurde. Matrizen aus reellen oder komplexen Zahlen sind dementsprechend die Grundelemente für alle Operationen.

1.1.6 Die Studentenversion von MATLAB

Die einzige Einschränkung in der Studentenversion von MATLAB ist die begrenzte Anzahl Elemente einer Matrix. Die maximale Grösse einer Matrix beträgt 16'384 Elemente. Die grösstmögliche quadratische Matrix hat somit 128 Kolonnen und 128 Zeilen. Die Studentenversion von MATLAB beinhaltet drei Toolboxes: die Signal Processing Toolbox, die Control System Toolbox und die Symbolic Math Toolbox.

1.2 Der Einstieg in MATLAB

Bemerkung: Alle Textausschnitte, die in diesem Bericht in der Schriftart **`Courier`** erscheinen, sind korrekte MATLAB-Ausdrücke, welche eingetippt in ein Command Window eine bestimmte Operation ausführen. Sie können sie mit Copy-Paste gleich verwenden, aber kopieren Sie den Prompt nicht mit!

1.2.1 Starten und Verlassen von MATLAB

Unter Unix geben Sie üblicherweise den Namen “matlab” in einem Command-Tool oder Xterm ein. Es wird zuerst für wenige Sekunden ein Splash-Window eingeblendet. Beim Erscheinen des Prompts befinden Sie sich mit dem Cursor in einem MATLAB Command Window. Falls der Befehl “matlab” nicht funktioniert, wenden Sie sich an Ihren Computeradministrator. Unter Macintosh und Windows können Sie MATLAB durch Doppelklicken der entsprechenden Ikone starten. Nach wenigen Sekunden erscheint ein Fenster, das Command Window. Befehle können Sie beim Erscheinen des Prompt-Zeichens “>>” (bzw. “EDU>>” bei der Studentenversion) eintippen.

Vergessen Sie nicht, Ihre Resultate (sowohl Daten als auch Graphiken) zu speichern. Beim Verlassen von MATLAB verlieren Sie sonst alle Ihren Daten! Um MATLAB zu verlassen, tippen Sie in das Command Window “quit” oder “exit” ein.

1.2.2 Das Command Window

In einem Command Window werden Befehle ausgeführt, welche Sie entweder gleich eintippen oder im voraus in einem M-File gespeichert haben. Zur Durchführung von Befehlabfolgen aus einem M-File tippen Sie normalerweise den M-Filenamen (ohne die Endung “.m”) in das Command Window ein. Achten Sie darauf, wie der Path in MATLAB gesetzt ist (-> `pwd`). Falls die auszuführenden M-Files sich in einem anderen Directory befinden, müssen Sie den Path neu setzen (-> `cd`). Die Ausgabe ist meistens im selben Command Window zu sehen oder, falls Sie Plot-Befehle eingegeben haben, in einem zwecks Darstellung der graphischen Resultate automatisch erschienenen Fenster. Graphische Fenster werden als “Figures” bezeichnet. Pro Plot-Befehl erscheint normalerweise ein Figure. Bei aufwendigen Berechnungen ist es von grossem Vorteil, wenn Sie die Befehlsabfolge in ein M-File schreiben, anstatt die einzelnen Befehle in das Command Window einzutippen.

1.2.3 Das M-File

M-Files sind nichts anderes als gewöhnliche Text-Files, deren Namen mit “.m” enden. Sie werden oft auch als “Scripts” bezeichnet. Befehle werden selten direkt in das Command Window eingetippt. Da die Befehle linienweise interpretiert werden, können Fehler kaum korrigiert werden. Gewöhnen Sie sich deswegen an, beim Starten einer Session gleich ein M-File aufzumachen. Die im M-File enthaltene Befehlabfolge können Sie dann wie oben beschrieben ausführen. Copy-Paste vom M-File zum Command Window hilft Ihnen, wenn Sie nur einige der Befehle ausführen möchten. In M-Files gespeicherte Befehlabfolgen können Sie jederzeit ändern, korrigieren oder erweitern. Testläufe von Programmen oder Teilen davon können somit effizient und zeitsparend erfolgen. Unter Unix können Sie ein beliebiges in ASCII gespeichertes File als Script verwenden. Um von MATLAB als solches erkannt zu werden, reicht es, dass es mit “.m” endet. Jeder Texteditor (u.a. emacs, vi, textedit) kann solche Files kreieren. Unter Macintosh und

Windows können Sie im MATLAB Menü unter “->File ->New M-File” ein M-File aufmachen. Natürlich können Sie auch unter Macintosh und Windows weitere Texteditoren verwenden. Achten Sie darauf, dass Sie keine reservierten Wörter (wie z.B. Namen von eingebauten Funktionen) als M-File-Namen benutzen. Die M-File-Namen sollten auch nicht mit dem Namen irgendwelcher Variablen übereinstimmen.

Sie können, und wegen der Lesbarkeit sollten Sie, Kommentare in Ihre Scripts einbetten. Das erreichen Sie, indem Sie ein Prozentzeichen “%” auf eine Zeile setzen. Der ganze Text, der dem Prozentzeichen auf derselben Zeile folgt, wird von MATLAB als Kommentar betrachtet und als solcher nicht interpretiert.

Sie können die Ausgabe von Resultaten im Command Window unterbinden, indem Sie am Ende einer Zeile einen Strichpunkt “;” anbringen. Die Befehle auf dieser Zeile werden trotzdem ausgeführt, und die Resultate können jederzeit abgefragt werden. Wenn Sie `disp(x)` in das Command Window eintippen, erscheint der Wert der Variable x ohne den Namen der Variable. Wenn Sie `disp('any text')` eintippen, erscheint “any text” im “Command Window”. Dieses Vorgehen eignet sich deshalb dazu, um sogenannte Flags in einem Programm anzubringen. Der Befehl `input('any text')` verlangt von Ihnen eine Eingabe über die Tastatur. Damit können Sie zum Beispiel Parameterwerte eingeben. Mit “pause” stoppen Sie die laufende Evaluation des M-Files, bis Sie eine beliebige Taste gedrückt haben. M-Files sind zudem notwendig, um neue Funktionen zu definieren. “Functions” sind i.a. in MATLAB programmierte Prozeduren, welche allfällige Argumente verarbeiten und/oder eine Reihe von Befehlen ausführen, um anschliessend ein Resultat zu liefern. Was Sie sich schon jetzt merken sollten ist, dass der Name der Funktion in der ersten Zeile eines M-Files mit dem Namen des M-Files übereinstimmen muss. Näheres erfahren Sie im Kapitel “Programmieren” dieses Kurses.

1.2.4 Die Figure

Mit dem Befehl “figure” öffnen Sie ein neues, für Graphiken reserviertes Fenster. Im Kapitel “Graphik” erfahren Sie mehr über das Plotten.

1.2.5 Die Definition von Variablen

Der Name einer Variablen darf fast beliebig gewählt werden. Folgende Regeln müssen Sie aber bei der Definition einer Variablen beachten: Der Variablenname darf nicht mit dem Namen eines M-Files übereinstimmen. Der Name muss ein einziges Wort ohne Leerzeichen sein, aus maximal 31 Zeichen bestehen und mit einem Buchstaben beginnen. Variablennamen dürfen Zahlen und Underscores “_” enthalten.

In MATLAB sind die Variablen von in M-Files definierten Funktionen lokal. Wenn z.B. der Variablenname “test” sowohl in einer Funktion als auch direkt im Command Window existiert, dann adressieren diese Namen zwei verschiedene Werte im Speicher. Sie können aber eine in einer Funktion definierte Variable global machen (d.h. sichtbar ausserhalb der Funktion), indem Sie sie explizit als “global” deklarieren.

1.3 Allgemeine Befehle

1.3.1 Erste Hilfe

Befehle	Kurzbeschreibung
`help`	On-line Hilfe
`demo`	MATLAB Demos

Befehl

`help`

Anwendung

On-line Hilfe im “Command Window” aufrufen.

Beschreibung

`help begriff` zeigt die zur Verfügung stehenden Kommentare im File “begriff.m”. Jede eingebaute Funktion in MATLAB beinhaltet einen Help-Abschnitt, welcher meistens aus einer ausführlichen Beschreibung der Befehlssyntax und oft mehreren Beispielen besteht.

`help` allein listet eine Reihe von Themenkreisen auf, aus welchen dann einzelne Begriffe abgefragt werden können.

Beispiel

```
>> help quit
 QUIT Quit MATLAB session.
      QUIT terminates MATLAB without saving work-
space. To save your workspace variables, use SAVE
before quitting.
      See also SAVE.
```

Siehe auch

`lookfor`, `dir`, `which`, `what`

Befehl

`demo`

Anwendung

MATLAB-eigenes Demonstrationsprogramm aufrufen.

Beschreibung

Mit `demo` wird das MATLAB-eigene Demonstrationsprogramm gestartet. Nebst einer kurzen Einführung beinhaltet es einige praktische Beispiele.

1.3.2 Handhabung des Workspaces

Befehle	Kurzbeschreibung
`who`	Alle Variablen auflisten
`whos`	Ausführliche Auflistung aller Variablen
`clear`	Alle lokalen Variablen löschen
`load`	Daten aus einer Datei laden
`save`	Daten in ein File speichern
`quit, exit`	MATLAB verlassen

Befehl

`who`

`whos`

Anwendung

Alle aktuellen Variablen auflisten.

Beschreibung

Mit `who` werden die Namen aller während der laufenden Session definierten Variablen aufgelistet. Mit `whos` werden ausführlichere Informationen gezeigt.

Beispiel

```
>> A=ones(2,2)
A =
      1     1
      1     1
>> whos A
      Name      Size         Bytes  Class
      A         2x2             32  double array
Grand total is 4 elements using 32 bytes
```

Siehe auch

`who global`

Befehl

`clear`

Anwendung

Variablen und Funktionen aus dem Speicher löschen.

Beschreibung

`clear` löscht alle vom Benutzer definierten Variablen.

Falls nur eine Variable “var” gelöscht werden muss, dann soll `clear var` verwendet werden.

`clear na*` löscht alle Variablen, deren Namen mit `na` beginnen.

Es ist ratsam, `clear` am Anfang jedes unabhängig laufenden M-Files einzuführen, um ältere Variablen aus dem Speicher zu entfernen.

Optionen

`clear all` entfernt alle Variablen, globalen Variablen, Funktionen und MEX-Verbindungen.

Beispiel

```
>> clear A
```

Siehe auch

`who`, `whos`

Befehl

`load`

Anwendung

Laden von Variablen oder Daten aus einem File oder einer Diskette in den Workspace.

Beschreibung

Mit `load filename` werden die in einem MAT-File namens “filename.mat” abgespeicherten Variablen in den Arbeitsspeicher eingelesen. Der ganze Path muss angegeben werden, falls das File sich nicht im aktuellen Directory befindet.

Auch Daten, die mit anderen Programmen (z.B. Tabellenkalkulation) erzeugt wurden, können eingelesen werden.

`load` alleine aktiviert die Variablen aus dem File “matlab.mat”.

`load filename X Y Z` lädt nur die Variablen X, Y und Z.

`load filename.ext` liest ASCII-Dateien, in denen kolonnenweise Daten gespeichert wurden.

Beispiel

```
>> % In einem Tabellenkalkulationsprogramm wurde
>> % eine Datei aus Messdaten unter dem Datei-
>> % namen/filename
>> % ‘test’ im text-format (ASCII-Format) ge-
>> % speichert.
>> % Diese Daten können in MATLAB wie folgt
>> % eingelesen werden:
>> load test
oder
>>load /home/user/Messdaten/test
```

Siehe auch

`save`, `whos`

Befehl

`save`

Anwendung

Speichert alle in der Arbeitsoberfläche definierten Variablen in ein File.

Beschreibung

Mit `save filename` werden alle definierten Variablen in einem MAT-File namens “filename.mat” abgespeicherten. Der ganze Path muss angegeben werden, falls das File nicht im aktuellen Directory gespeichert werden soll.

`save filename X` speichert nur die Variable X ab.

Beispiel

```
>> save test
>> save test zeitvektor
>> save('test','zeitvektor')
```

Siehe auch

`load`

Befehl

`quit`

`exit`

Anwendung

MATLAB beenden.

Beschreibung

Mit `quit` oder `exit` wird MATLAB verlassen. Die definierten Variablen werden nicht automatisch gespeichert.

Siehe auch

`save`

1.3.3 MATLAB Funktionen suchen und editieren

Befehle	Kurzbeschreibung
`edit`	M-File öffnen
`lookfor`	M-Files nach Schlüsselwörtern absuchen
`which`	Lokalisieren von M-Files

Befehl

`edit`

Anwendung

M-File öffnen.

Beschreibung

Mit `edit filename` wird vom default Texteditor das File filename aufgemacht. Der ganze Path muss angegeben werden, falls das File sich nicht im aktuellen Directory befindet.

Um unter Unix den default Texteditor zu definieren, muss – bevor MATLAB gestartet wird – in eine shell z.B. “setenv EDITOR textedit” oder “setenv EDITOR xemacs” eingetippt werden.

Befehl

`lookfor`

Anwendung

Alle M-Files nach Schlüsselwörtern absuchen.

Beschreibung

Mit `lookfor string` wird in allen zur Verfügung stehenden M-Files und in allen Beschreibungen von eingebauten Funktionen nach der Zeichenfolge “string” gesucht. Eine Liste von möglicherweise zutreffenden Begriffen wird anschliessend im Command Window editiert. `lookfor` eignet sich zur Suche von Funktionen, wenn nur eine Vorstellung vorhanden ist, welche Begriffe, Wörter oder Wortabschnitte in Beschreibungsteil vorhanden sein könnten. Achten Sie darauf, dass alle MATLAB-Helps auf englisch geschrieben sind!

Beispiel

```
>> % Wie lautet der Correlations-Befehl
>> % im MATLAB?
>> % Vermutlich enthält die Beschreibung eines
>> % solchen Befehls die Zeichenfolge “corr”.
>> lookfor corr
CORRCOEF Correlation coefficients.
SFUNCORR  an S-function which performs auto- and
cross-correlation.
TDER  Time-Delay Estimation using ML windowed
cross-correlation
D_TDER  HOSA Demo: Time-delay estimation using
cross-correlation (tder)
CRA  Performs correlation analysis to estimate
impulse response.
IDUICRA  Handles the correlation analysis dialog.
CORR2  Compute 2-D correlation coefficient.
XCORR  Cross-correlation function estimates.
XCORR2  Two-dimensional cross-correlation.
```

Siehe auch

`dir`, `help`, `who`, `which`, `what`

Befehl

`which`

Anwendung

Kurzbeschreibung und Lokalisierung von Variablen, Funktionen und M-Files.

Beschreibung

Mit `which filename` wird der Path des M-Files filename gezeigt. Mit `which xyz` wird beschrieben, was für ein Objekt xyz ist (ob Variable, eingebaute Funktion usw.).

Beispiel

```
>> which A
A is a variable.
>> which sin
sin is a built-in function.
>> which cov
/usr/local/src/MATLAB5/toolbox/MATLAB/datafun/
cov.m
```

Siehe auch

`dir`, `help`, `who`, `lookfor`

1.3.4 Ausgabeformat

Befehle	Kurzbeschreibung
`format`	Ausgabeformat wählen

Befehl

`format`

Anwendung

Das Ausgabeformat definieren.

Beschreibung

Mit `format` wird das Ausgabeformat von numerischen Resultaten definiert: Fliesskommaformat mit 4 oder 14 Nachkommastellen oder wissenschaftliche Notation mit 5- oder 16-stelliger Mantissa und 2-stelligem Exponenten.

Optionen

```
>> format short
>> pi
ans =
      3.1416
>> format long
ans =
      3.14159265358979
>> format short e
ans =
      3.1416e+00
>> format long e
ans =
      3.141592653589793e+00
```


1.3.5 Systembefehle

Befehle	Kurzbeschreibung
`cd`	Den Ordner wechseln
`pwd`	Den aktuellen Pfad anzeigen
`dir`	Inhaltsliste des aktuellen Ordners anzeigen
`delete`	Datei oder Graphik löschen

Befehl

`cd`

Anwendung

Änderung des laufenden Directory.

Beschreibung

Mit `cd path` wird der Path gesetzt und damit das laufende Directory geändert.

Unter Unix ist normalerweise das laufende Directory das Home-Directory des Benutzers. `cd ..` springt in das übergeordnete Directory, `cd` allein zeigt den aktuellen Path.

Siehe auch

`pwd`

Befehl

`pwd`

Anwendung

Zeigt den aktuellen Path.

Siehe auch

`cd`

Befehl

`dir`

Anwendung

Directories und Files im aktuellen Directory auflisten.

Beschreibung

`dir` listet alle Directories und Files im aktuellen Directory auf.

`dir *.m` listet alle M-Files im aktuellen Directory auf.

Beispiel

```
>> cd
/home/muster/MATLAB
>> dir
messdaten
test1.m
test2.m
```

Siehe auch

`cd`, `delete`, `what`

Befehl

`delete`

Anwendung

Löschen von Files.

Beschreibung

`delete filename` löscht das File namens filename.

`delete *.m` löscht alle M-files im aktuellen Directory.

Beispiel

```
>> cd
/home/muster/MATLAB
>> cd messdaten
>> cd
/home/muster/MATLAB/messdaten
>> dir
Motor_1.m
Motor_2.m
>> delete Motor_2.m
```


1.3.6 Spezielle Tasten

Befehle	Kurzbeschreibung
`ctrl-P`	In der Command History nach oben springen
`ctrl-N`	In der Command History nach unten springen
`ctrl-B`	Eine Stelle nach links springen
`ctrl-F`	Eine Stelle nach rechts springen
`ctrl-A`	An den Zeilenanfang springen
`ctrl-E`	An das Zeilenende springen
`ctrl-U`	Eine Zeile löschen
`ctrl-C`	Laufende Rechnung stoppen

Befehl

`ctrl-P`

`ctrl-N`

Anwendung

Schnellere Eingabe von schon eingetippten Befehlen.

Beschreibung

MATLAB führt eine Liste der während einer Session durchgeführten Befehle. Mit `ctrl-P` (für Previous) und `ctrl-N` (für Next) werden die Befehle in der Liste der Reihe nach am Prompt gezeigt.

Befehl

`ctrl-B`

`ctrl-F`

Anwendung

Änderung von schon eingetippten Befehlen.

Beschreibung

Mit `ctrl-B` (für Backward) und `ctrl-F` (für Forward) läuft der Cursor rück- bzw. vorwärts über eine Zeile. Damit können Änderungen an einem schon eingetippten Befehl durchgeführt werden, ohne dass der Befehl gelöscht werden muss.

Befehl

`ctrl-A`

`ctrl-E`

Anwendung

Mit dem Cursor an den Zeilenanfang bzw. das Zeilenende springen.

Befehl

`ctrl-U`

Anwendung

Eine Zeile löschen.

Befehl

`ctrl-C`

Anwendung

Eine laufende Rechnung unterbrechen.

Beschreibung

Mit `ctrl-C` wird die Durchführung eines Befehls gestoppt.

1.4 Spezielle Zeichen und Operatoren

1.4.1 Spezielle Zeichen

Befehle	Kurzbeschreibung
`()`	Runde Klammern für Argumente
`[]`	Eckige Klammern für Matrizen
`.`	Punkt für Dezimalstellen
`..`	Ins obere Directory wechseln
`...`	Zeilen verbinden
`,`	Trennzeichen für Kolonnen in einer Matrix
`;`	Trennzeichen für Zeilen
`'`	Halbe Anführungszeichen für einen Text
`%`	Anbringen von Kommentaren
`=`	Wertzuweisung

Befehl

`( )`

Anwendung

Bei Befehlen und Funktionen die Argumente definieren.

Auf Elemente eines Vektors oder einer Matrix zugreifen.

Festlegung der Priorität der Rechenoperationen

Beschreibung

Befehle und Funktionen, die Daten in Form von Vektoren, Matrizen oder Strings benötigen, sind mit einem Klammerpaar `( )` zu versehen (z.B. `plot(x,y,'c')`).

Wurde eine n×n Matrix `A` im Workspace definiert, so greift der Befehl `A(m,n)` auf das n-te Element der m-ten Zeile dieser Matrix zu. Einige hilfreiche Matrixmanipulationen sind im Beispiel aufgeführt.

Runde Klammern werden wie in der Analysis üblich für die Festlegung der Priorität der Rechenoperationen verwendet.

Beispiel

```
>> sqrt(9)
ans =
     3
>> mtimes(3,3)
ans =
     9
>> disp('Klammern im MATLAB')
Klammern im MATLAB
>> A=[1 2 3;4 5 6;7 8 9];
A =
     1     2     3
     4     5     6
     7     8     9
>> A(3,1)
ans =
     7
>> A(3,1)=0
A =
     1     2     3
     4     5     6
     0     8     9
>> A(:,1)=[1;1;1]
A =
     1     2     3
     1     5     6
     1     8     9
>> A(:,1)=[]
A =
     2     3
     5     6
     8     9
>> 1/(sqrt(9)+1)
ans =
     0.2500
>> 1/sqrt(9)+1
ans =
     1.3333
```


Befehl

`[ ]`

Anwendung

Vektoren und Matrizen definieren.

Beschreibung

Mit dem eckigen Klammerpaar `[ ]` (Tastenkombination alt-5 und alt-6 bei Mac und PC) werden Vektoren und Matrizen definiert. Jede Zeile wird mit einem Strichpunkt ; abgeschlossen, die Kolonnen mit einem Komma , oder einer Leertaste.

Beispiel

```
>> A=[1 2 3]
A =
     1     2     3
>> B=[1;2;3]
B =
     1
     2
     3
>> C=[1 2 3;4 5 6;7 8 9]
C =
     1     2     3
     4     5     6
     7     8     9
>> A*B
ans =
     14
>> B*A
ans =
     1     2     3
     2     4     6
     3     6     9
```

Befehl

`.`

Anwendung

Dezimalstelle bei rationalen Zahlen

Elementweise Vektoren und Matrizen multiplizieren und dividieren.

Beschreibung

Mit einem Punkt `.` werden die Dezimalstellen einer rationalen Zahl angegeben.

Ein Punkt vor den Operatoren `*` , `\` , `/` und `^` bewirkt, dass die Multiplikation, die Links- und Rechtsdivision und die Exponentialfunktion elementweise abläuft.

Beispiel

```
>> A=[1.5 2.75;3.1 1.125]
A =
      1.5000    2.7500
      3.1000    1.1250
>> pi
ans =
      3.1416
>> B=[3.25 1.8;2.1 4.6]
B =
      3.2500    1.8000
      2.1000    4.6000
>> A.*B
ans =
      4.8750    4.9500
      6.5100    5.1750
```

Siehe auch

`format`, `.*`

Befehl

`..`

Anwendung

Mit `cd ..` in das nächst obere Directory wechseln.

Beschreibung

Zwei Punkte `..` werden nur in Verbindung mit dem Befehl `cd` gebraucht. Mit dem Befehl `cd ..` wechselt der Path vom aktuellen Directory in das hierarchisch nächst höherliegende Directory.

Beispeil

```
>> cd
/home/muster/MATLAB/messdaten
>> dir
Motor_1.m
Motor_2.m
>> cd ..
>> cd
/home/muster/MATLAB
>> dir
messdaten
test1.m
test2.m
```

Siehe auch

`cd`

Befehl

`...`

Anwendung

Zeilen verbinden.

Beschreibung

Drei Punkte `...` am Ende einer Zeile verbinden das Ende der Zeile mit dem Anfang der folgenden Zeile. Für eine übersichtliche Darstellung werden lange Zeilen oft mit drei Punkten zwei- oder dreigeteilt.

Beispiel

```
>> A=[1 12 23 34 45 56 67 78 89 910 111 ...
212 313 414 515 616 717 818 919 120]
A =
      Columns 1 through 12
       1    12    23    34    45    56    67    78
89   910   111   212
      Columns 13 through 20
       313   414   515   616   717   818   919
120
```


Befehl

`,`

Anwendung

Trennzeichen für Kolonnen

Trennen von mehreren Eingabeargumenten in Befehlen und Funktionen.

Beschreibung

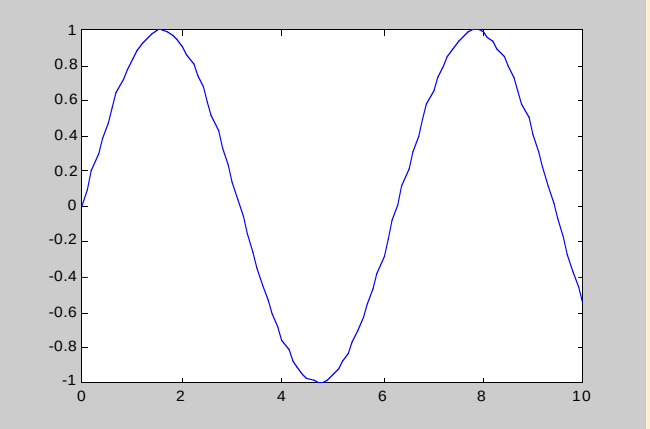
Bei der Definition von Matrizen werden die Kolonnen durch Kommas oder Leerzeichen getrennt.

Befehle oder Funktionen können mehrere Eingabeargumente haben. Diese werden mit Kommas voneinander getrennt.

Mehrere Befehle auf einer Zeile werden ebenfalls mit Kommas getrennt. Dies sollte jedoch vermieden werden.

Beispiel

```
>> A=[1,2,3;4,5,6;7,8,9]
A =
      1     2     3
      4     5     6
      7     8     9
>> t=(0:0.1:10);
>> plot(t,sin(t))
```



Befehl

`;`

Anwendung

Trennzeichen für Zeilen

Ausgabe im Command Window unterdrücken, die normalerweise auf eine Eingabe folgen würde.

Beschreibung

Bei der Definition von Matrizen werden die Zeilen mit einem Strichpunkt `;` abgeschlossen.

Die Ausgabe von Zwischenresultaten wird im Command Window unterdrückt, wenn die entsprechende Befehlzeile mit einem Strichpunkt abgeschlossen wurde.

Beispiel

```
>> A=[1 2 3;4 5 6;7 8 9];
>> B=[1 0 1;0 1 0;1 0 1];
>> A*B
ans =
       4     2     4
      10     5    10
      16     8    16
```


Befehl

`'`

Anwendung

Mit halben Anführungszeichen einen beliebigen Text als Vektor definieren.

Beschreibung

Buchstaben und Zeichen können innerhalb von zwei halben Anführungszeichen als Zeichenreihe dargestellt werden.

`'Ein beliebiger Text'` ist ein Vektor. Jedes einzelnen Zeichen dieses Vektors wird intern im entsprechenden ASCII-Code abgespeichert.

Ein halbes Anführungszeichen innerhalb eines beliebigen Textes wird mit zwei halben Anführungszeichen definiert.

Beispiel

```
>> 'Text zwischen zwei halben Anführungszeichen'
ans =
Text zwischen zwei halben Anführungszeichen
>> whos
  Name      Size         Bytes  Class
  ans       1x43            86  char array
Grand total is 43 elements using 86 bytes
>> class(ans)
ans =
char
>> 'Ich hab''s kapiert!'
ans =
Ich hab's kapiert!
```

Befehl

%

Anwendung

Kommentare anbringen.

Beschreibung

Programme sollten ausführliche Kommentare enthalten, damit das Programmierte zu einem späteren Zeitpunkt leichter nachvollzogen werden kann.

Programme ohne Kommentare sind nutzlos.

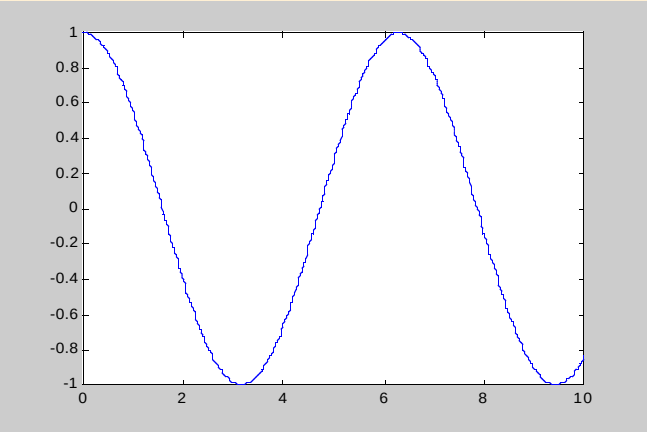
Ein Kommentar besteht aus einem Prozentzeichen % und einem darauffolgenden Text.

Es ist ratsam, ein M-File mit einer kurzen Inhaltsangabe hinter Prozentzeichen % zu versehen.

Insbesondere sollte vor jedem einzelnen Programm-Abschnitt ein kurzer Kommentar stehen.

Beispiel

```
>> % Sinus-Plot
>> t=(0:0.01:10); % Vektor mit einem Inkrement...
von 0.1 zwischen 0 und 10
>> plot(t,cos(t)) % Sinus-Plot mit der Variable t
```



Befehl

=

Anwendung

Wert zuweisen.

Beschreibung

Das Gleichheitszeichen weist einer Variable einen bestimmten Zahlenwert zu.

Beispiel

```
>> A=6;
>> A
A =
      6
>> B=A;
>> B
B =
      6
>> A
A =
      6
```

1.4.2 Arithmetische Operatoren

Befehle	Kurzbeschreibung
+	Addition
-	Subtraktion
*	Matrix Multiplikation
.*	Elementweise Multiplikation
\	Linke Matrix Division
/	Rechte Matrix Division
^	Exponentialfunktion

Befehl

+

-

Anwendung

Addition und Subtraktion

Beschreibung

Das + –Zeichen addiert Zahlen, Vektoren oder Matrizen. Bei der Addition von Vektoren oder Matrizen müssen die Dimensionen übereinstimmen.

- ist das Zeichen für die Subtraktion von Zahlen, Vektoren oder Matrizen. Die Matrixdimensionen müssen bei der Subtraktion ebenfalls übereinstimmen.

Beispiel

```
>> 7+12-3
ans =
      16
>> plus(6,25)
ans =
      31
```


Befehl

*

.*

Anwendung

Matrix- und Vektormultiplikation

Beschreibung

Mit * werden Matrizen oder Vektoren miteinander multipliziert. Dabei ist zu beachten, dass bei der Multiplikation zweier Vektoren oder Matrizen A*B die Anzahl Kolonnen von A mit der Anzahl Zeilen von B übereinstimmen. Falls die Dimensionen nicht übereinstimmen, erscheint im Command Window die Fehlermeldung: “Inner matrix dimensions must agree”.

.* multipliziert Vektoren oder Matrizen elementweise. Die Anzahl Zeilen und Kolonnen von A und B müssen identisch sein.

Beispiel

```
>> A=[1 2 3; 4 5 6];
>> B=[1 2; 3 4;5 6];
>> A*B
ans =
      22    28
      49    64
>> B*A
ans =
       9    12    15
      19    26    33
      29    40    51
>> A.*A
ans =
       1     4     9
      16    25    36
```

Befehl

`\`

`/`

Anwendung

Linke und rechte Division

Beschreibung

Die linke Division wird für das Lösen von linearen Gleichungssystemen verwendet.

Für das lineare Gleichungssystem in der Form **Ax = b** lautet die nach x aufgelöste Gleichung $x = A^{-1}b$. Dafür wird der Befehl `A\b` verwendet. A ist eine quadratische Matrix und b ein Kolonnenvektor.

Falls das lineare Gleichungssystem die Form **xA = b** hat, kommt die rechte Division zur Anwendung.

Die Lösung ist $x = bA^{-1}$, und der dazugehörige MATLAB-Befehl lautet `b/A`.

Wie bei der Multiplikation können die Vektoren und Matrizen elementweise dividiert werden. Vor das Divisionszeichen ist dafür ein Punkt zu setzen.

Beispiel

```
>> A=[1 0 1;0 1 1;1 1 0]
A =
      1     0     1
      0     1     1
      1     1     0
>> b=[1;1;1]
b =
      1
      1
      1
>> A\b
ans =
      0.5000
      0.5000
      0.5000
```


Befehl

`^`

Anwendung

Matrix Exponentialfunktion

Beschreibung

`^` kann auf quadratische Matrizen angewandt werden, um A^n zu berechnen. `n` ist eine Integer. Für n>0 wird die Matrix A n mal mit sich selbst multipliziert. n = 0 ergibt die Identitätsmatrix. n<0 entspricht der Inversen von A.

Beispiel

```
>> A=[1 2;0 1]
A =
      1     2
      0     1
>> A^2
ans =
      1     4
      0     1
>> A^-2
ans =
      1    -4
      0     1
>> inv(A)
ans =
      1    -2
      0     1
>> inv(A)^2
ans =
      1    -4
      0     1
```

1.4.3 Logische Operatoren

Befehle	Kurzbeschreibung
&	Logisches und
|	Logisches oder
~	Logisches nicht
==	Gleichheitszeichen
<=	kleiner gleich
>=	grösser gleich
<	kleiner
>	grösser

Befehl

&

Anwendung

Logisches “und”

Beschreibung

Das logische “und” vergleicht zwei gleich grosse Objekte miteinander. Der logische Vergleich erfolgt elementweise. Somit sind auch Vektoren und Matrizen untereinander vergleichbar.

Haben die Matrizen A und B dieselbe Grösse, so ergibt sich aus A&B wiederum eine Matrix derselben Grösse mit Nullen und Einsen.

Ist sowohl das eine Element von A als auch das entsprechende von B ungleich Null, so erhält die Anwort-Matrix an derjenigen Stelle eine Eins. Ist irgendein Element von A oder B gleich Null, dann schreibt MATLAB an derselben Stelle eine Null.

Beispiel

```
>> A=[4 0 2 -3 0 1];
>> B=[2 1 0 5 0 4];
>> A&B
ans =
     1     0     0     1     0     1
```


Befehl

|

Anwendung

Logisches “oder”

Beschreibung

Für das logische “oder” gelten die gleichen Vorschriften wie für das logische “und”. Ein Element der resultierenden Matrix ist jedoch nur dann Null, falls das Element von A und das entsprechende von B ebenfalls Null sind.

Beispiel

```
>> A=[4 0 2 -3 0 1];
>> B=[2 1 0 5 0 4];
>> A|B
ans =
     1     1     1     1     0     1
```

Befehl

~

Anwendung

Logisches “nicht”

Beschreibung

Für das logische “nicht” gelten dieselben Vorschriften wie für das logische “und”. Der Befehl ~A wandelt alle Elemente der Matrix A, die den Wert Null haben, in eine Eins und alle anderen in eine Null um.

Beispiel

```
>> A=[4 0 2 -3 0 1];
>> ~A
ans =
     0     1     0     0     1     0
```

Befehl

==

Anwendung

Gleichheitszeichen

Beschreibung

Das doppelte Gleichheitszeichen == vergleicht zwei Matrizen miteinander. Es kann aber auch ein Vektor und eine Matrix oder ein Skalar und ein Vektor gegenübergestellt werden. Der Vektor wird mit den einzelnen Zeilen oder Kolonnen der Matrix verglichen und der Skalar mit den einzelnen Elementen des Vektors. Dieser Befehl wird häufig bei der Programmierung von “while” Schlaufen, “if-else-end” Beziehungen oder “switch-case” Konstruktionen verwendet.

Aufgepasst! Ein einziges Gleichheitszeichen = weist einer Variablen einen bestimmten Wert zu.

Beispiel

```
>> m=1;
>> if m==0
disp('Die Variable m ist 0')
else disp('Die Variable m ist nicht 0')
end
ans =
Die Variable m ist nicht 0
```


Befehl

<=

>=

Anwendung

Kleiner gleich oder grösser gleich

Beschreibung

Sowohl in “for” und “while” Schlaufen als auch in “if-else-end” und “switch-case” Konstruktionen kommen <= und >= zur Anwendung. Sie werden oft als Abbruchkriterium benützt. Solange im Vergleich die Bedingung “grösser gleich” oder “kleiner gleich” wahr ist, wird die Schlaufe durchlaufen.

Beispiel

```
>> n=0;
>> while n<=2
disp(n)
n=n+1;
end

     0
     1
     2
```

Befehl

`<`

`>`

Anwendung

Grösser oder kleiner

Beschreibung

`<` und `>` werden wie `<=` und `>=` verwendet. Gleichheit liefert aber die Anwort ‘falsch’ bzw. den Wert 0.

Beispiel

```
>> n=0;
>> while n<2
disp(n)
n=n+1;
end

     0
     1
```


1.5 Elementare Mathematik

1.5.1 Trigonometrie

Befehle	Kurzbeschreibung
`sin`	Sinus
`cos`	Cosinus
`tan`	Tangens
`cot`	Cotangens

Befehl

`sin`

Anwendung

Sinus-Funktion

Beschreibung

`sin(x)` berechnet den Sinus von x in Radians. x ist eine beliebige Zahl.

Beispiel

```
>> sin(pi)
ans =
      1.2246e-16
>> sin(pi/2)
ans =
      1
```

Befehl

`cos`

Anwendung

Cosinus-Funktion

Beschreibung

`cos(x)` berechnet den Cosinus von x in Radians. x ist eine beliebige Zahl.

Beispiel

```
>> cos(pi)
ans =
      -1
>> cos(pi/2)
ans =
      6.1232e-17
```

Befehl

`tan`

Anwendung

Tangens-Funktion

Beschreibung

`tan(x)` berechnet den Tangens von x in Radians. x ist eine beliebige Zahl.

Beispiel

```
>> tan(0)
ans =
      0
>> tan(pi/4)
ans =
      1.0000
>> tan(pi/2)
ans =
      1.6331e+16
```


Befehl

`cot`

Anwendung

Cotangens-Funktion

Beschreibung

`cot(x)` berechnet den Cotangens von x in Radians. x ist eine beliebige Zahl.

Beispiel

```
>> cot(0)
Warning: Divide by zero.
ans =
      Inf
>> cot(pi/4)
ans =
      1.0000
>> cot(pi/2)
ans =
      6.1232e-17
```

1.5.2 Exponential- und Logarithmusfunktionen

Befehle	Kurzbeschreibung
`exp`	e-Funktion
`log`	Natürlicher Logarithmus
`sqrt`	Quadratwurzel

Befehl

`exp`

Anwendung

Exponentialfunktion

Beschreibung

`exp(x)` berechnet die Exponentialfunktion von x, d.h. e^x.

Beispiel

```
>> exp(1)
ans =
      2.7183
>> exp(100)
ans =
      2.6881e+43
```

Siehe auch

`expm`

Befehl

`log`

Anwendung

Natürlicher Logarithmus

Beschreibung

`log(x)` berechnet den natürlichen Logarithmus von x, d.h. ln(x).

Beispiel

```
>> log(1)
ans =
      0
>> log(0)
Warning: Log of zero.
ans =
      -Inf
```

Siehe auch

`logm`

Befehl

`sqrt`

Anwendung

Quadratwurzel

Beschreibung

`sqrt(x)` berechnet die Quadratwurzel von x, d.h. $\sqrt{x}$.

Beispiel

```
>> sqrt(2)
ans =
      1.4142
>> sqrt(-1)
ans =
      0 + 1.0000i
```

Siehe auch

`sqrtm`

1.5.3 Komplexe Zahlen

Befehle	Kurzbeschreibung
`abs`	Absolutwert einer Zahl
`angle`	Phasenwinkel
`conj`	konjugiert komplexe Zahl
`imag`	Imaginärteil
`real`	Realteil

Befehl

`abs`

Anwendung

Betrag einer Zahl

Beschreibung

`abs(x)` berechnet den Absolutwert von x, d.h. |x|.

Falls x eine komplexe Zahl ist, ermittelt `abs(x)` den Betrag von x.

Beispiel

```
>> abs(-pi)
ans =
      3.1416
>> abs(1-i)
ans =
      1.4142
```

Siehe auch

`sign`

Befehl

`angle`

Anwendung

Berechnung der Phase

Beschreibung

`angle(x)` berechnet den Winkel von x in Radians, wobei x eine komplexe Zahl ist. Die komplexe Zahl besteht aus einem Real- und einem Imaginärteil y = x+iy. Der Winkel ergibt sich aus dem Tangens von Imaginär- über Realteil: φ = tan(y/x).

Beispiel

```
>> angle(1)
ans =
      0
>> angle(sqrt(-1))
ans =
      1.5708
>> ans*180/pi % in Grad
ans =
      90
```

Befehl

`conj`

Anwendung

Konjugiert komplexe Zahl

Beschreibung

`conj(z)` berechnet die Konjugiertkomplexe der Zahl z, d.h. $\bar{z}$.

Beispiel

```
>> sqrt(-1)
ans =
      0 + 1.0000i
>> conj(sqrt(-1))
ans =
      0 - 1.0000i
```

Siehe auch

`i`

Befehl
`imag`

Anwendung
Imaginärteil einer komplexen Zahl

Beschreibung
`imag(z)` gibt den Imaginärteil der komplexen Zahl z wieder.

Beispiel
```
>> imag(sqrt(-1))
ans =
      1
```

Befehl
`real`

Anwendung
Realteil einer komplexen Zahl

Beschreibung
`real(z)` gibt den Realteil der komplexen Zahl z wieder.

Beispiel
```
>> real(sqrt(-1))
ans =
      0
```


1.5.4 Runden

Befehle	Kurzbeschreibung
`fix`	Runden in Richtung 0
`floor`	Abrunden
`ceil`	Aufrunden
`round`	Runden
`sign`	Vorzeichen bestimmen

Befehl
`fix`

Anwendung
Runden in Richtung Null.

Beschreibung
`fix(x)` rundet in Richtung Null auf die nächste ganze Zahl.

Beispiel
```
>> fix(1.99)
ans =
      1
>> fix(-1.99)
ans =
      -1
```

Befehl

`floor`

Anwendung

Auf den nächstkleineren ganzzahligen Wert runden.

Beschreibung

`floor(x)` rundet die natürliche Zahl x auf den nächstkleineren ganzzahligen Wert.

Beispiel

```
>> floor(1.99)
ans =
      1
>> floor(-1.01)
ans =
      -2
```

Befehl

`ceil`

Anwendung

Auf den nächstgrösseren ganzzahligen Wert runden.

Beschreibung

`ceil(x)` rundet die natürliche Zahl x auf den nächstgrösseren ganzzahligen Wert.

Beispiel

```
>> ceil(1.1)
ans =
      2
>> ceil(-1.9)
ans =
      -1
```


Befehl

`round`

Anwendung

Auf den nächstliegenden ganzzahligen Wert runden.

Beschreibung

`round(x)` rundet die Zahl x auf den nächstliegenden ganzzahligen Wert.

Beispiel

```
>> round(1.5)
ans =
      2
>> round(1.49)
ans =
      1
```

Befehl

`sign`

Anwendung

Vorzeichenbestimmung

Beschreibung

`sign(x)` gibt das Vorzeichen von x an.

Beispiel

```
>> sign(2)
ans =
      1
>> sign(0)
ans =
      0
>> sign(-2)
ans =
      -1
```

2 Vektoren und Matrizen

Die Grundstruktur von MATLAB ist eine n×n-Matrix aus reellen und/oder komplexen Elementen.

$$A = \begin{bmatrix} a_{11} + i \cdot c_{11} & a_{12} + i \cdot c_{12} & \dots & a_{1m} + i \cdot c_{1m} \\ a_{21} + i \cdot c_{21} & a_{22} + i \cdot c_{22} & \dots & a_{2m} + i \cdot c_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} + i \cdot c_{n1} & a_{n2} + i \cdot c_{n2} & \dots & a_{nm} + i \cdot c_{nm} \end{bmatrix} \quad (1)$$

Der erste Index steht für die Zeile, der zweite für die Kolonne.

Eine typische Anwendung von Matrizen ist das Lösen eines linearen Gleichungssystems **Ax = b**.

$$\begin{bmatrix} a_{11} + i \cdot c_{11} & a_{12} + i \cdot c_{12} & \dots & a_{1m} + i \cdot c_{1m} \\ a_{21} + i \cdot c_{21} & a_{22} + i \cdot c_{22} & \dots & a_{2m} + i \cdot c_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} + i \cdot c_{n1} & a_{n2} + i \cdot c_{n2} & \dots & a_{nm} + i \cdot c_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} \quad (2)$$

Spezialfälle sind Vektoren (n×1- oder 1×n-Matrizen) und Skalare (1×1-Matrix).

Dieses Kapitel behandelt die elementaren Matrizen und die Matrixmanipulationen.

2.1 Elementare Matrix-Manipulationen

2.1.1 Elementare Matrizen

Befehle	Kurzbeschreibung
`zeros`	Matrix mit lauter Nullen
`ones`	Matrix mit lauter Einsen
`eye`	Identitätsmatrix
`diag`	Diagonalmatix
`rand`	Matrix mit lauter Zufallszahlen
`linspace`	Linear unterteilter Vektor
`meshgrid`	Matrix aus zwei Vektoren

Befehl

`zeros`

Anwendung

Alle Koeffizienten einer Matrix sind Null.

Beschreibung

`zeros` setzt alle Koeffizienten einer Matrix gleich Null.

`zeros(n)` bildet eine n×n Matrix, deren Elemente alle den Wert Null besitzen.

`zeros(n,m)` steht für eine Rechtecksmatrix mit n Zeilen und m Kolonnen mit lauter Nullen.

[Bemerkung: MATLAB kennt das symbolische Rechnen nicht. Jeder Variable muss ein numerischer Wert zugewiesen werden. Mit Hilfe von `zeros` kann z.B. eine Matrix definiert werden, bei der am Anfang alle Elemente Null sind.**]**

Beispiel

```
>> zeros(2)
ans =
     0     0
     0     0
```

Befehl

`ones`

Anwendung

Alle Koeffizienten einer Matrix haben den Betrag Eins.

Beschreibung

`ones` weist allen Koeffizienten einer Matrix den Wert Eins zu.

`ones(n)` bildet eine quadratische Matrix, deren Elemente alle den Betrag Eins haben. Die Matrix `ones(n,m)` besteht aus n Zeilen und m Kolonnen.

Jeder Variable muss ein numerischer Wert zugewiesen werden.

Beispiel

```
>> ones(2)
ans =
     1     1
     1     1
>> ones(2,3)
ans =
     1     1     1
     1     1     1
>> A+B
??? Undefined function or variable 'A'.
>> A=zeros(2);
>> B=ones(2);
>> A(1,1)=1;
>> A(1,2)=5;
>> A(2,2)=-3;
>> A
A =
     1     5
     0    -3
>> A+B
ans =
     2     6
     1    -2
```


Befehl

`eye`

Anwendung

Alle Diagonalelemente einer Matrix haben den Betrag Eins.

Beschreibung

`eye(n)` bildet die Identitätsmatrix in $R^{n \times n}$. Sie ist in quadratischer Form.

`eye(n,m)` definiert eine Rechtecksmatrix mit n Zeilen und m Kolonnen. Die Diagonalelemente haben den Betrag Eins.

Beispiel

```
>> eye(2)
ans =
     1     0
     0     1
>> ones(2,3)
ans =
     1     0     0
     0     1     0
>> A=[1 2;3 4];
>> eye(size(A))
ans =
     1     0
     0     1
```

Siehe auch

`ones`, `zeros`, `rand`

Befehl

`diag`

Anwendung

Diagonalmatrix bilden.

Beschreibung

Mit `diag(a)` wird der Vektor a in der Diagonale einer quadratischen Matrix eingebettet. Die Koeffizienten ausserhalb der Diagonale sind Null.

`diag(A)` bildet die Diagonale einer beliebigen Matrix in einem Kolonnenvektor ab.

Beispiel

```
>> a=[1 2 3 4];
>> diag(a)
ans =
     1     0     0     0
     0     2     0     0
     0     0     3     0
     0     0     0     4
>> A=[1 2 3;4 5 6]
A =
     1     2     3
     4     5     6
>> diag(A)
ans =
     1
     5
```

Siehe auch

`triu`, `tril`

Befehl

`rand`

Anwendung

Die Koeffizienten einer Matrix bestehen aus gleichmässig verteilten Zufallszahlen.

Beschreibung

`rand(n)` weist allen Koeffizienten einer Matrix in $R^{n \times n}$ eine gleichmässig verteilte Zufallszahl zwischen Null und Eins zu.

`rand(n,m)` hat n Zeilen und m Kolonnen mit n×m gleichmässig verteilten Zufallszahlen.

`rand(n,m,p)` generiert p Matrizen in $R^{n \times m}$.

Beispiel

```
>> rand(2)
ans =
      0.9501    0.6068
    0.2311    0.4860
>> rand(3,4)
ans =
      0.8381    0.3795    0.7095    0.1897
      0.0196    0.8318    0.4289    0.1934
      0.6813    0.5028    0.3046    0.6822
>> rand(2,2,2)
ans(:,:,1) =
      0.8214    0.6154
      0.4447    0.7919

ans(:,:,2) =
      0.9218    0.1763
      0.7382    0.4057
```

Siehe auch

`randn`, `randperm`

Befehl

`linspace`

Anwendung

Vektor in $R^{1\times m}$ mit konstantem Abstand zwischen den Koeffizienten.

Beschreibung

`linspace(xstart, xend)` erzeugt einen Vektor zwischen xstart und xend, der in 99 gleiche Intervalle unterteilt wird. Der Vektor besteht somit aus 100 linear, gleichmässig verteilten Punkten.

`linspace(xstart,xend,n)` bildet einen Vektor mit n linerar unterteilten Punkten zwischen xstart und xend.

Beispiel

```
>> linspace(0,10,11)
ans =
     0    1    2    3    4    5    6    7
8    9   10
```

Siehe auch

`logspace`

Befehl

`meshgrid`

Anwendung

Aus zwei oder drei Vektoren Matrizen in der Form eines zwei- oder dreidimensionalen Gitters bilden.

Beschreibung

`[X,Y]=meshgrid(x,y)` formt aus den Vektoren $x \in R^m$ und $y \in R^n$ die Matrizen X und Y mit je n×m Elementen. X und Y werden aus den Vektoren x und y gebildet, indem x in n Zeilen und y in m Kolonnen kopiert werden. Die Matrizen X und Y werden für das Plotten von Funktionen mit zwei Variablen und für dreidimensionale “Oberflächen” Graphiken verwendet.

`[X,Y,Z]=meshgrid(x,y,z)` berechnet auf dieselbe Art ein dreidimensionales Gitter. Die Matrizen X,Y und Z werden für das Plotten von Funktionen mit drei Variablen und für dreidimensionale “Volumen” Graphiken gebraucht.

Beispiel

```
>> x=[0 1 0];
>> y=[0 0.5 1];
>> [X,Y]=meshgrid(x,y)
X =
     0     1     0
     0     1     0
     0     1     0
Y =
          0         0         0
     0.5000    0.5000    0.5000
     1.0000    1.0000    1.0000
```

Siehe auch

`surf`

2.1.2 Information über die Dimension

Befehle	Kurzbeschreibung
`size`	Dimension einer Matrix
`length`	Dimension eines Vektors
`disp`	Zeichenfolge im Command Window editieren

Befehl

`size`

Anwendung

Angabe über die Anzahl Zeilen und Kolonnen einer Matrix.

Beschreibung

`size(A)` informiert über die Dimension der Matrix A. Die erste Zahl des zweizeiligen Ausgabevektors steht für die Anzahl Zeilen von A, die zweite für die Anzahl Kolonnen.

Mit `[M,N]=size(A)` werden die Zeilen- und Kolonnenzahl von A den Variablen M und N zugewiesen.

`size(A,1)` gibt Auskunft über die Anzahl Zeilen der Matrix A und `size(A,2)` über die Anzahl Kolonnen.

Beispiel

```
>> A=[1 2;3 4; 5 6];
>> size(A)
ans =
     3     2
>> [M,N]=size(A)
M =
     3
N =
     2
>> size(A,1)
ans =
     3
```

Siehe auch

`length`

Befehl

`length`

Anwendung

Angabe über die Länge eines Vektors.

Beschreibung

`length(a)` ermittelt die Anzahl Zeilen des Kolonnenvektors a bzw. die Anzahl Kolonnen des Zeilenvektors a.

Beispiel

```
>> a=[1 2 3 4]
a =
     1     2     3     4
>> length(a)
ans =
     4
>> a=[1;2;3]
a =
     1
     2
     3
ans =
     3
```

Befehl

`disp`

Anwendung

Wiedergabe einer Zeichenfolge (String oder Matrix) im Command Window, ohne den Namen der Zeichenfolge anzugeben.

Beschreibung

`disp(X)` editiert im Command Window die in X definierte Zeilenfolge. Dabei wird der Namen der Zeilenfolge bei der Ausgabe unterdrückt.

Ist X ein String, so erscheint ein beliebiger Text im Command Window. Damit können z.B. in M-Files Kontrollpunkte eingefügt werden. Sobald der Rechner im M-File eine bestimmte Teilaufgabe erfolgreich gelöst hat, kann dies mit `disp('text')` im Command Window angezeigt werden.

Bei gewissen Berechnungen benötigt ein M-File fortlaufend Input aus dem Command Window. Für diese Kommunikation zwischen M-File und Command Window eignen sich die Befehle `disp` und `input`.

Beispiel

```
>> A=[1 2;3 4];
>> disp(A)
     1     2
     3     4
>> B=inv(A); % Die Inverse von A berechnen
>> disp(['Die Matrix A wurde invertiert',...
  ' und unter der Variable B abge',...
  'speichert.'])
Die Matrix A wurde invertiert und unter der Vari-
able B abgespeichert.
>> B
B =
     -2.0000    1.0000
      1.5000   -0.5000
```

Siehe auch

`format`

2.1.3 Spezielle Variablen und Konstanten

Befehle	Kurzbeschreibung
`ans`	default-Namen für die letzte Ausgabe
`eps`	Fliesskomma bezüglich der Genauigkeit
`realmax`	Grösstmögliche Fliesskomma-Zahl
`pi`	Zahl π
`i`	i = $\sqrt{-1}$ Imaginärteil einer Zahl
`inf`	Unendlich

Befehl

`ans`

Anwendung

default-Variablennamen für das Resultat der letzten Berechnung.

Beschreibung

Die zuletzt berechnete Ausgabe wird der Variable `ans` zugewiesen, falls kein anderer Namen definiert wurde.

Beispiel

```
>> sqrt(24)*5^2.5
ans =
      273.8613
```

Befehl

`eps`

Anwendung

Rechengenauigkeit des Computers.

Beschreibung

`eps` gibt den Abstand an, der zwischen der Zahl Eins und der nächstmöglichen Fliesskomma-Stelle liegt. Es ist der kleinste Wert, der zur Zahl Eins addiert werden kann, damit sich die Summe von Eins unterscheidet.

Diese Toleranz kann geändert werden. Dabei ist zu beachten, dass mit dem Befehl `clear` der neu definierte Wert für `eps` nicht auf den ursprünglichen Betrag zurückspringt.

Beispiel

```
>> eps
ans =
      2.2204e-16
>> 1==1+eps % Frage: Ist 1 gleich 1+eps?
ans =
      0 % 1 = ja; 0 = nein
```

Siehe auch

`realmax`, `realmin`

Befehl

`realmax`

Anwendung

Grösstmögliche Fliesskomma-Zahl des Computers

Beschreibung

Mit `realmax` eruiert MATLAB die grösstmögliche positive reelle Zahl, die der Computer berechnen kann.

Beispiel

```
>> realmax
ans =
      1.7977e+308
```

Siehe auch

`eps`, `realmin`

Befehl

`pi`

Anwendung

Zahl π

Beschreibung

Mit `pi` berechnet MATLAB die Zahl π. π ist das Verhältnis zwischen dem Kreisumfang und dem Kreisdurchmesser.

Falls `eps` genügend klein ist, ermittelt MATLAB für `pi` bis zu 16 Dezimalstellen.

Beispiel

```
>>pi
ans =
      3.1416
>> format long
>> pi
ans =
      3.14159265358979
```

Befehl

`i`

Anwendung

Imaginärteil einer komplexen Zahl

Beschreibung

`i` oder `j` stehen für dem Imaginärteil einer komplexen Zahl und haben den Wert $\sqrt{-1}$.

Beispiel

```
>> 1-2i
ans =
      1.0000- 2.0000i
>> 1-2*i
ans =
      1.0000- 2.0000i
>> 1-2*sqrt(-1)
ans =
      1.0000- 2.0000i
```

Siehe auch

`j`, Komplexe Zahlen

Befehl

`inf`

Anwendung

Unendlicher Wert

Beschreibung

MATLAB antwortet mit dem Ausdruck `inf`, falls ein Wert gegen unendlich geht (z.B. Division mit der Zahl Null im Nenner).

Strebt während einer Berechnung ein bestimmter Wert gegen unendlich, so wird dies im Command Window angezeigt.

Beispiel

```
>> 1/0
Warning: Divide by zero.
ans =
      ∞
>> 1/0;1+1
Warning: Divide by zero.
ans =
      2
```

Siehe auch

`NaN`

2.2 Lineare Algebra

2.2.1 Grundoperationen

Befehle	Kurzbeschreibung
`/ and \`	Rechte und linke Matrix Division
`'`	Transponierte einer Matrix
`inv`	Inverse einer Matrix
`cond`	Kondition einer Matrix
`rank`	Rang einer Matrix
`norm`	Norm einer Matrix oder eines Vektors
`det`	Determinante
`trace`	Spur einer Matrix
`orth`	Orthonormierte Basisvektoren einer Matrix

Befehl

`/ and \`

Anwendung

Rechte Matrix Division / und linke Matrix Division \

Beschreibung

`/` und `\` stehen für die rechte und linke Matrix Division. Die linke Division von `A\B` entspricht der Multiplikation der Inversen der regulären Matrix A (regulär => $\det(A) \neq 0$) mit der Matrix B, d.h. `inv(A)*B`. Dasselbe gilt für A/B und B*inv(A). `A\b` ist die Lösung des linearen Gleichungssystems Ax = b, falls $A \in R^{n \times n}$ und Rang(A) = n. Bei einer Rechtecksmatrix $A \in R^{n \times m}$ mit n>m handelt es sich um ein überbestimmtes Gleichungssystem. Das System hat keine exakte Lösung. Mit `A\b` liefert MATLAB jene approximierte Lösung, für die der totale Fehler e für alle n Gleichungen am kleinsten ist. Dieser entspricht im MATLAB der Summe aller Fehler im Quadrat:

$$e = \sum_{i=1}^{n} \left(b_i - \sum_{j=1}^{m} a_{i,j} x_j \right)^2 = \| b - Ax \|_2^{\,2} \qquad (3)$$

Für ein unterbestimmtes Gleichungssystem n<m gibt es unendlich viele Lösungen. MATLAB sucht sich selber eine aus.

Beispiel

```
>> A=[1 0 2;2 1 1;0 1 1];
>> B=[1 0 3;4 2 1;1 0 1];
>> A\B
ans =
       1.5000    1.0000         0
       1.2500    0.5000   -0.5000
      -0.2500   -0.5000    1.5000
>> A/B
ans =
      0.5000         0    0.5000
      0.2500    0.5000   -0.2500
      1.2500    0.5000   -3.2500
>> b=[1;0;1];
>> A\b
ans =
      -0.5000
       0.2500
       0.7500
>> A=[1 0;2 1;0 1];
>> A\b
ans =
      0.0000
      0.5000
>> A=[1 0 2;2 1 1];
>> b=[1;0];
>> A\b
ans =
      -0.3333
            0
       0.6667
```

Siehe auch

`slash`

Befehl

`'`

Anwendung

Transponierte einer reellen oder komplexen Matrix

Beschreibung

$[a_{ij}]$ sei eine reelle Matrix $A \in R^{n\times m}$. Ihre Transponierte ist definiert durch $A^T = [a_{ji}]$. In MATLAB wird das halbe Anführungszeichen ' für die Transponierte A' der Matrix A verwendet.

Ist A symmetrische, so gilt $A^T = A$ und in MATLAB `A' = A`.

Ist $[a_{ij}]$ eine komplexe Matrix $A \in C^{n\times m}$, so lautet ihre konjugiert-transponierte Matrix $A^H = [\bar{a}_{ji}]$ bzw. A' in MATLAB.

`A.'` ist der Befehl für die nicht-konjugiert-transponierte Matrix $A^T = [a_{ji}]$ einer komplexen Matrix $A \in C^{n\times m}$.

Beispiel

```
>> A=[1 2 3;4 5 6]
A =
      1     2     3
      4     5     6
>> A'
ans =
      1     4
      2     5
      3     6
>> B=[1+i 2-2i;3+3i 4-4i]
B =
      1.0000+ 1.0000i   2.0000- 2.0000i
      3.0000+ 3.0000i   4.0000- 4.0000i
>> B'
ans =
      1.0000- 1.0000i   3.0000- 3.0000i
      2.0000+ 2.0000i   4.0000+ 4.0000i
>> B.'
ans =
      1.0000+ 1.0000i   3.0000+ 3.0000i
      2.0000- 2.0000i   4.0000- 4.0000i
```

Befehl

`inv`

Anwendung

Inverse einer quadratischen, regulären Matrix

Beschreibung

`inv(A)` berechnet die inverse Matrix der quadratischen, regulären Matrix A. Die quadratische Matrix heisst regulär, wenn $\det(A) \neq 0$ ist. Falls sie singulär ist, erscheint im Command Window von MATLAB eine Fehlermeldung. Die Definition der inversen Matrix von A lautet:

$$X = A^{-1} = \frac{1}{\det(A)}[(-1)^{i+j}\det(A_{ji})] \qquad (4)$$

Beispiel

```
>> A=[1 0 1;1 1 2;0 2 1]
A =
     1     0     1
     1     1     2
     0     2     1
>> inv(A)
ans =
      3    -2     1
      1    -1     1
     -2     2    -1
>> A=[1 0 1;1 1 2;0 2 2]
A =
     1     0     1
     1     1     2
     0     2     2
>> inv(A)
Warning: Matrix is singular to working precision.
ans =
     ∞     ∞     ∞
     ∞     ∞     ∞
     ∞     ∞     ∞
```

Siehe auch

`slash`, `cond`

Befehl

`cond`

Anwendung

Die Kondition einer Matrix

Beschreibung

`cond(A)` gibt mit einer reellen Zahl Auskunft über die Kondition des Systems, das durch Matrix A beschrieben wird. Sie ist grösser oder gleich Eins, und misst die Empfindlichkeit der Lösung x auf Störungen im linearen Gleichungssystem Ax = b. Die Kondition ist das Verhältnis zwischen dem grössten und dem kleinsten Singularwert einer Matrix. Die Singularwerte σ von A lassen sich mit der Euklidischen Norm berechnen. Sie sind die positiven Quadratwurzeln der grössten Eigenwerte der Hermiteschen Matrix $A^H A$:

$$\sigma_i(A) = \sqrt{\lambda_i(A^H A)}. \qquad (5)$$

A^H ist die konjugiert-transponierte Matrix von A, falls $A \in C^{n \times n}$.

Bei regulären Matrizen ist der kleinste Singularwert immer grösser Null. Singuläre Matrizen sind nicht invertierbar, und haben einen Singularwert bei Null. Die Kondition einer singulären Matix geht somit gegen unendlich. Eine grosse Zahl zeigt an, dass sich die betreffende Matrix nahe bei einer singulären Matrix befindet, d.h. das System ist schlecht konditioniert.

Beispiel

```
>> A=[1 0 1;1 1 2;0 2 1];
>> cond(A)
ans =
      16.3937
>> B=[1 0 1;1 1 2;0 2 2];
>> cond(B)
ans =
      ∞
```

Siehe auch

`norm`

Befehl

`rank`

Anwendung

Rang einer Matrix

Beschreibung

`rank(A)` berechnet die Anzahl linear unabhängiger Zeilen oder Kolonnen der Matrix A.

Beispiel

```
>> A=[1 2 3;4 5 6];
A =
     1     2     3
     4     5     6
>> rank(A)
ans =
     2
>> B=[1 0 2;1 1 0;2 0 4]
B =
     1     0     2
     1     1     0
     2     0     4
>> rank(B)
ans =
     2
```


Befehl

`norm`

Anwendung

Die unterschiedlichen Normen eines Vektors oder einer Matrix

Beschreibung

Die Norm eines Vektors ist ein Skalar. Er misst die Grösse bzw. die Länge eines Vektors.

`norm(a)` berechnet die Euklidische Norm

$$\|a\|_2 = \sqrt{\sum_k |a_k|^2}. \tag{6}$$

Dies entspricht `sum(abs(a).^2)^(1/2)`.

`norm(a,1)` berechnet die Summe der absoluten Beträge von jedem Vektorelement, `sum(abs((a)))`.

$$\|a\|_1 = \sum_k |a_k| \tag{7}$$

`norm(a,inf)` ermittelt die ∞-Norm von a. Sie steht für das betragsmässig grösste Zeilenelement von a, `max(abs(a))`.

Die Norm einer Matrix misst die Grösse bzw. den Betrag einer Matrix.

`norm(A)` eruiert die Euklidische Norm der Matrix A. Sie entspricht dem grössten Singularwert von A, `max(svd(A))`.

`norm(A,1)` berechnet die betragsmässig grösste Summe der absoluten Beträge von jedem Kolonnenelement von A, `max(sum(abs((A))))`.

`norm(A,inf)` ermittelt die betragsmässig grösste Summe der absoluten Beträge von jedem Zeilenelemente von A, `max(sum(abs((A'))))`.

Beispiel

```
>> a=[1 -2 3];A=[1 2;-3 4];
>> norm(a)
ans =
      3.7417
>> norm(a,1)
ans =
      6
>> norm(a,inf)
ans =
      3
>> norm(A)
ans =
      5.1167
>> norm(A,inf)
ans =
      7
```

Siehe auch

cond

Befehl

det

Anwendung

Determinante einer quadratischen Matrix

Beschreibung

Für eine Matrix $A \in R^{n \times n}$, wobei $n > 1$ ist, gilt für die Determinante von A folgende Definition:

$$\det(A) = \sum_{k=1}^{n} (-1)^{j+k} a_{kj} \det(A_{kj}). \qquad (8)$$

Der Buchstabe j steht hier für die j-te Kolonne.

Beispiel

```
>> A=[1 2;1 4]
A =
      1     2
      1     4
>> det(A)
ans =
      2
>> B=[1 0 2;2 1 1;0 1 1]
B =
      1     0     2
      2     1     1
      0     1     1
ans =
      4
```

Siehe auch

cond

Befehl

`trace`

Anwendung

Spur einer Matrix bzw. Summe der Diagonalelemente

Beschreibung

`trace(A)` ermittelt die Summe der Diagonalelemente der Matrix A. Sie wird auch Spur einer Matrix genannt und entspricht der Summe der Eigenwerte von A.

Beispiel

```
>> A=[1 0 3;0 2 4]
A =
      1     0     3
      0     2     4
>> trace(A)
ans =
      3
>> B=[1 2 0;0 5 2;7 3 9]
B =
      1     2     0
      0     5     2
      7     3     9
>> trace(B)
ans =
      15
>> sum(eig(B))
ans =
      15
```


Befehl

`orth`

Anwendung

Orthonormierte Basisvektoren einer Matrix

Beschreibung

`Q=orth(A)` eruiert die orthonormierte Basismatrix von A. Sie spannt denselben Raum auf wie die Matrix A. Die Anzahl Kolonnen der orthonormierten Basismatrix entspricht dem Rang der Matrix A.

Q^TQ ergibt die Identitätsmatrix I.

Beispiel

```
>> A=[1 3;3 9]
A =
      1     3
      3     9
>> orth(A)
Q =
      0.3162
      0.9487
>> Q'*Q % Q transponiert multipliziert mit Q
ans =
      1
>> B=[5 7;9 2]
B =
      5     7
      9     2
>> Q=orth(B)
Q =
    0.6735    0.7392
    0.7392   -0.6735
>> Q'*Q
ans =
      1.0000    0.0000
      0.0000    1.0000
```

Siehe auch

`rank`, `null`

2.2.2 Eigenwerte und Singularwerte

Befehle	Kurzbeschreibung
`eig`	Eigenwerte und Eigenvektoren einer Matrix
`svd`	Singularwerte einer Matrix

Befehl

`eig`

Anwendung

Eigenwerte und Eigenvektoren einer Marix berechnen.

Beschreibung

Die Eigenwerte λ und die dazu gehörenden Eigenvektoren x einer Matrix $A \in R^{n \times n}$ lassen sich aus folgendem linearen Gleichungssystem berechnen:

$$(Ax_i = \lambda_i x_i) \text{ bzw. } (\lambda_i I - A)x_i = 0 \in R^n. \quad (9)$$

λ ist ein Skalar und steht bei einer n×n Matrix für die n Eigenwerte.

$$(\lambda_i I - A)x_i = 0 \quad (10)$$

(10) hat genau dann eine nichttriviale Lösung, wenn (11)

$$\det(\lambda_i I - A) \quad (11)$$

gleich Null ist. xi ist Element des Nullraumes:

$$x_i \in N(\lambda_i I - A) \text{ bzw. } \left\{ x_i \in R^n \middle| (\lambda_i I - A)x_i = 0 \in R^n \right\} \quad (12)$$

Sowohl die Eigenwerte als auch die n Eigenvektoren können reell oder komplex sein.

`eig(A)` berechnet die Eigenwerte der Matrix A.

`[eigv, eigw]=eig(A)` weist der Variable eigv die Eigenvektoren von A zu. In der Diagonalmatrix eigw befinden sich die dazugehörenden Eigenwerte.

Beispiel

```
>> A=[1 3 2;0 4 2;1 -1 -2]
A =
      1     3     2
      0     4     2
      1    -1    -2
>> [eigv,eigw]=eig(A)
eigv =
     -0.8890   -0.3015   -0.7071
      0.2540   -0.3015   -0.7071
     -0.3810    0.9045   -0.0000
eigw =
      1.0000         0         0
           0   -2.0000         0
           0         0    4.0000
>> B=[-1 2 1;2 1 0;2 -4 1]
B =
     -1     2     1
      2     1     0
      2    -4     1
>> [eigv,eigw]=eig(B)
eigv =
     -0.6667   0.0647- 0.2697i   0.0647+ 0.2697i
      0.3333  -0.2050- 0.3344i  -0.2050+ 0.3344i
      0.6667   0.8738- 0.0756i   0.8738+ 0.0756i
eigw =
     -3.0000        0                  0
      0        2.0000+ 1.0000i         0
      0             0            2.0000- 1.0000i
```

Siehe auch

`condeig`

Befehl

`svd`

Anwendung

Singularwerte einer Matrix berechnen.

Beschreibung

`svd(A)` ermittelt alle Singularwerte der Matrix A. Wie bereits beim Befehl `cond` erklärt wurde, sind die Singularwerte σ der Matrix A die positiven Quadratwurzeln der grössten Eigenwerte der Hermiteschen Matrix $A^H A$.

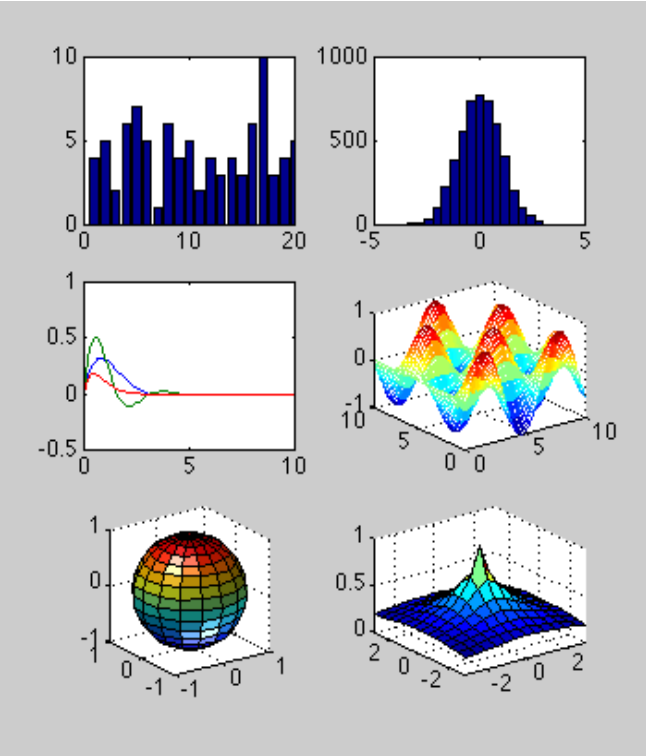
Die Singularwerte geben an, um welchen Faktor sich ein Vektor x durch die Abbildung mit der Matrix A in der Länge und der Richtung verändert. Sie charakterisieren das “Verstärkungsverhalten” einer beliebigen Matrix.

Beispiel

```
>> A=[1 2 1;2 2 4;1 0 5];
>> B=[1 2 1;2 2 4;1 0 1];
>> svd(A)
ans =
      7.0651
      2.4557
      0.2305
>> svd(B)
ans =
      5.4659
      1.3535
      0.5407
>> cond(A)
ans =
      30.6452
>> cond(B)
ans =
      10.1091
```


3 Graphik

MATLAB besitzt eine Vielzahl von Befehlen, um numerische Daten ansprechend graphisch darzustellen.



In diesem Kapitel beschreiben wir die wichtigsten Befehle zur Erzeugung graphischer Darstellungen. Es gibt neben diesen viele weitere Befehle, mit denen eine Graphik noch verfeinert werden kann. Es ist typisch, dass diese Verfeinerungen dann einen grossen Teil des Programmes ausmachen werden und nicht etwa die Definitionen der aufzuzeichnenden Funktionen.

3.1 Zweidimensionale Graphik

3.1.1 Elementare zweidimensionale Graphik

Befehle	Kurzbeschreibung
`plot`	2-D-Graphik mit linearen Achsenskalierungen
`subplot`	Mehrere 2-D-Graphiken in einem Fenster
`semilogx`	2-D-Graphik mit einfachlogarithmischer Achsenskalierung
`loglog`	2-D-Graphik mit doppeltlogarithmischer Achsenskalierung
`polar`	2-D-Graphik in Polarkoordinaten
`plotyy`	2-D-Graphik mit zwei Ordinaten

Befehl

`plot`

Anwendung

Zweidimensionale Graphik mit linearen Achsenskalierungen

Mehrere zweidimensionale Graphiken in einem Fenster

Beschreibung

Der Befehl `plot` öffnet ein Graphikfenster namens “figure” mit einer Nummer, in das eine Graphik eingebettet werden kann. Falls für die Abszisse und für die Ordinate keine Schranken gesetzt werden, passt sich die Skalierung des Koordinatensystems den Daten automatisch an (autoscaling). Wichtig: Für jedes weitere Bild muss mit dem Befehl `figure` ein neues Graphikfenster geöffnet werden, es erhält eine neue Nummer. Andernfalls wird das alte Bild im Graphikfenster durch das neue Bild überschrieben.

Bekanntlich basiert die Grundstruktur von MATLAB auf einer n×m-Matrix aus reellen oder komplexen Elementen (siehe “Wie ist MATLAB aufgebaut?”). Auch Daten werden ja in Matrizen abgelegt (Wertetabelle). Für ein zweidimensionales Bild benötigt MATLAB also mindestens zwei Kolonnenvektoren gleicher Länge.

Der Befehl `plot(x,y)` zeichnet den Datensatz y in Funktion von Datensatz x auf. Wie üblich, wird jedem Wert von x ein Wert von y zugeordnet. x sind die Werte auf der Abszisse und y diejenigen auf der Ordinate. Die daraus resultierenden Punkte werden mit geraden Linien verbunden (lineare Interpolation). Beide Achsen haben eine lineare Skala.

Mit `plot(x,y,s)` werden im String s der Linientyp und die Farbe der Kurve definiert. `help plot` listet im Command Window eine Auswahl von möglichen Liniendefinitionen auf. `plot(x,y,'c+:')` plottet z.B. eine rot punktierte Linie, die bei jedem Datenpunkt (x,y) ein rotes Plus-Zeichen hat.

Der Befehl `plot(y)` enthält nur einen Kolonnenvektor y Element von $R^{n\times1}$. In diesem Fall generiert MATLAB für die x-Achse automatisch Werte, nämlich 1 bis n, die Indizes der n Kolonnenwerte.

Handelt es sich jedoch bei y um einen Vektor mit komplexen Zahlen, so werden beim Befehl `plot(y)` die Realteile auf der x-Achse und die Imaginärteile auf der y-Achse aufgetragen.

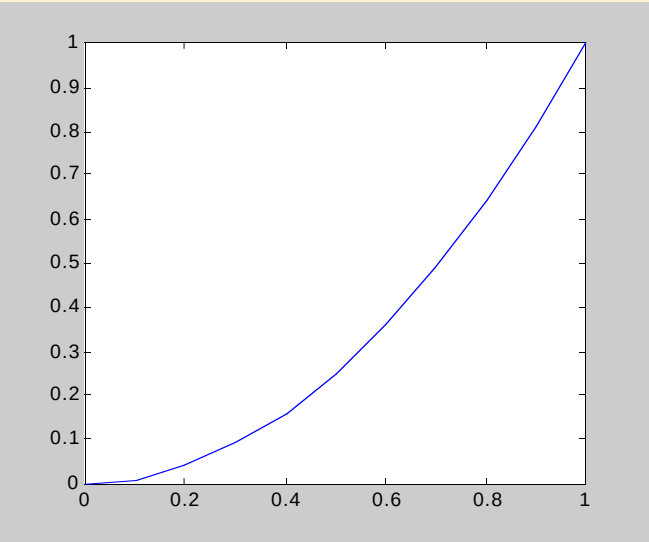
Der Befehl `plot(A)` zeichnet für jede der m Kolonnen der Matrix A Element von $R^{n\times m}$ je eine Kurve aus n Punkten. Die x-Achse zeigt wieder die Indizes 1…n. Die Linien werden zur Unterscheidung verschiedene Stile beziehungsweise verschiedene Farben haben.

Beim Befehl `plot(x,A)` wird jede der m Kolonnen der Matrix A Element von $R^{n\times m}$ gegen die gemeinsame unabhängige Variable x aufgezeichnet. Der Vektor x muss die Dimension n haben: x Element von $R^{n\times1}$.

Beim Befehl `plot(A,B)` bilden je eine Kolonne der Matrix A Element von $R^{n\times m}$ und der Matrix B Element von $R^{n\times m}$ ein x-, y-Vektorpaar.

Beispiel

```
>> figure % Graphikfenster wird vorbereitet
>> % unaghängige Variable wird festgelegt
>> % es entstehen 11 Datenpunkten
>> x=0:0.1:1;
>> % aghängige Variable wird definiert (Funktion)
>> % es entstehen ebenfalls 11 Datenpunkten
>> y=x.^2; % abhängige Variable
>> plot(x,y)
```



Siehe auch

`grid`, `clf`, `clc`, `title`, `xlabel`, `ylabel`, `axis`, `hold`, `subplot`

Befehl

`subplot`

Anwendung

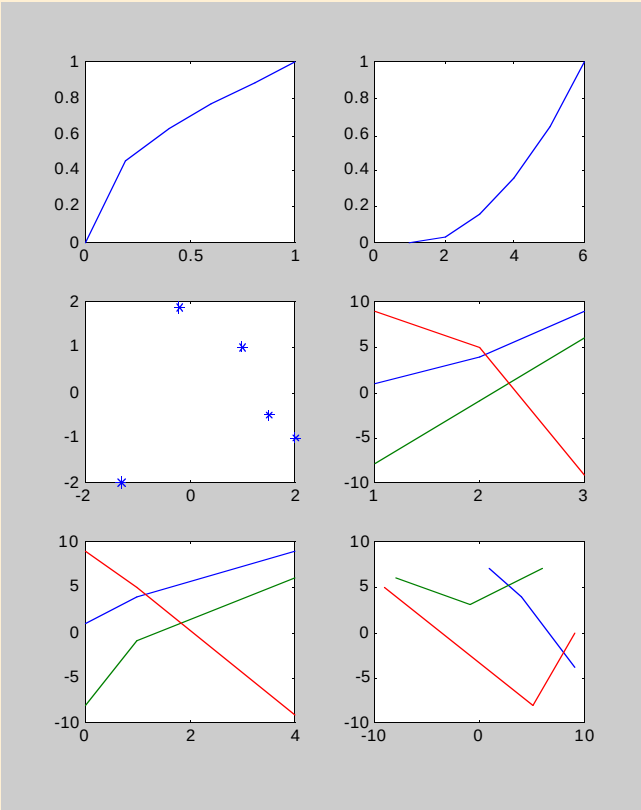
Mehrere zweidimensionale Graphiken in einem Fenster

Beschreibung

Der Befehl `subplot(n,m,p)` unterteilt ein Graphikfenster in n Zeilen von je m Bildern. Damit können n×m Bilder in ein Graphikfenster eingebettet werden (siehe Titelbild dieses Kapitels). p ist der Laufindex der n×m Bilder, wobei die Numerierung zeilenweise von links nach rechts erfolgt. Für jedes neue Bild im Graphikfenster wird der Befehl `subplot` wiederholt, jedesmal mit dem neuen Index p. Der eigentliche Befehl `plot` mit seinen Parametern muss dann natürlich auch noch kommen.

Beispiel

```
>> figure
>> % 3x2=6 Bilder in einem Fenster
>> subplot(3,2,1) % Anweisungen zu Bild 1
>> x=[0:0.2:1]; % unabhängige Variable
>> plot(x,sqrt(x)) % Wurzel von x
>> subplot(3,2,2) % Anweisungen zu Bild 2
>> % Datenvektor ohne unabhängige Variable x
>> % MATLAB erzeugt eigenen Indexvektor 1 bis 6
>> y=[0 0.04 0.16 0.36 0.64 1];
>> plot(y)
>> subplot(3,2,3) % Anweisungen zu Bild 3
>> % Vektor komplexer Zahlen in der Gauss-Ebene
>> z=[1+i;2-i;1.5-0.5i;-0.2+1.9i;-1.3-2i];
>> % es werden Sterne * anstelle von Punkten .
>> % gezeichnet
>> plot(z,'*')
>> subplot(3,2,4) % Anweisungen zu Bild 4
>> % 3 Datenvektoren mit je 3 Werten
>> A=[1  -8  9;4 -1 5;9 6 -9];
>> % Index 1 bis 3 von x wird durch MATLAB erzeugt
>> plot(A)
>> subplot(3,2,5) % Anweisungen zu Bild 5
>> % es werden Werte für eine unabhängige
>> % Variable x vorgegeben; die alten Werte von x
>> % sind bisher immer überschrieben worden.
>> x=[0 1 4];
>> plot(x,A)
>> subplot(3,2,6) % Anweisungen zu Bild 6
>> B=[7 6 0;4 3 -8;-4 7 5];
>> plot(A,B)
```

Befehl

`semilogx, semilogy`

Anwendung

Zweidimensionale Graphik mit logarithmischer x-Achse

Beschreibung

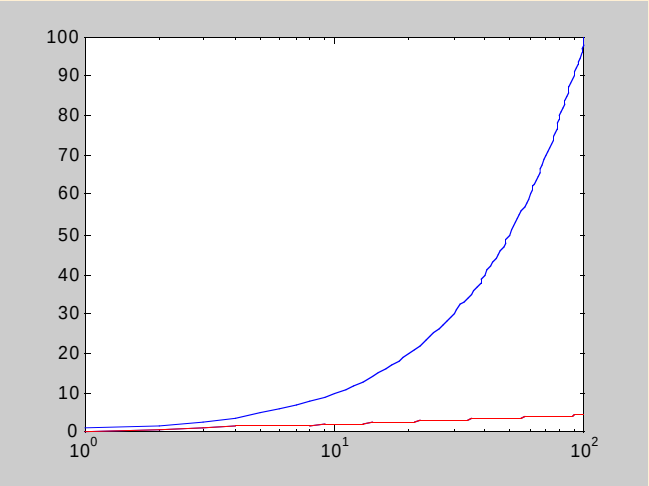
Die Befehle `semilogx` und `plot` sind bis auf die Skalierung der x-Achse identisch. Beim Befehl `plot` hat die Abszisse eine lineare, beim Befehl `semilogx` aber eine logarithmische Skala (Basis 10).

Der Befehl `semilogx(x,y)` entspricht dem Befehl `plot(log10(x),y)`, doch MATLAB gibt bei semi`logx` für x = 0 keine Warnung “log of zero”. Der Nullpunkt der x-Achse wird unterdrückt, er kann nicht gezeichnet werden.

Mit dem Befehl `semilogy` wird die Ordinate mit dem Zehnerlogarithmus skaliert.

Beispiel

```
>> x=[0:1:100];
>> semilogx(x,x) % Funktion y(x)=x (Gerade)
>> hold on
>> semilogx(x,exp(x))% Funktion y(x)=e^x
```



Siehe auch

`semilogy`

Befehl

`loglog`

Anwendung

Zweidimensionale Graphik über doppeltlogarithmisch skalierten Achsen

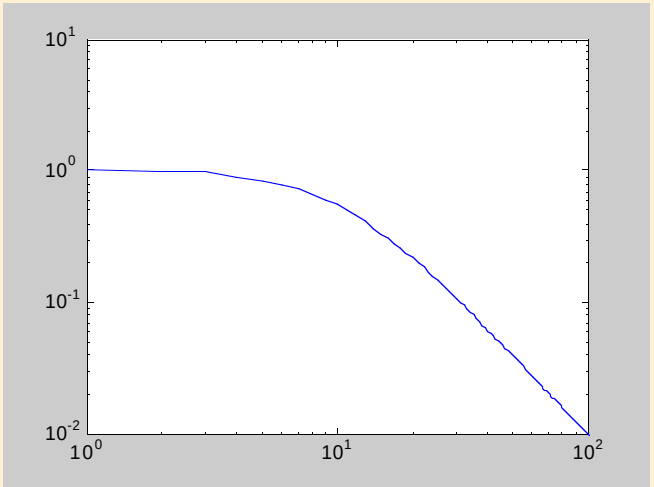
Beschreibung

Der Befehl `loglog` plottet eine zweidimensionale Graphik in einem doppeltlogarithmischen Koordinatensystem (Basis 10).

Der Befehl `loglog(x,y), log10(y))` entspricht dem Befehl `plot(log10(x))`, doch MATLAB gibt bei `loglog` keine Warnung “log of zero”, falls x oder y gleich Null ist. Der Koordinatennullpunkt wird unterdrückt.

Beispiel

```
>> x=[0:0.1:100]; % x zw. 0 und 100
>> loglog(1./(1+x.^2))
>> axis([0 100 0.01 10])
```



Befehl

`polar`

Anwendung

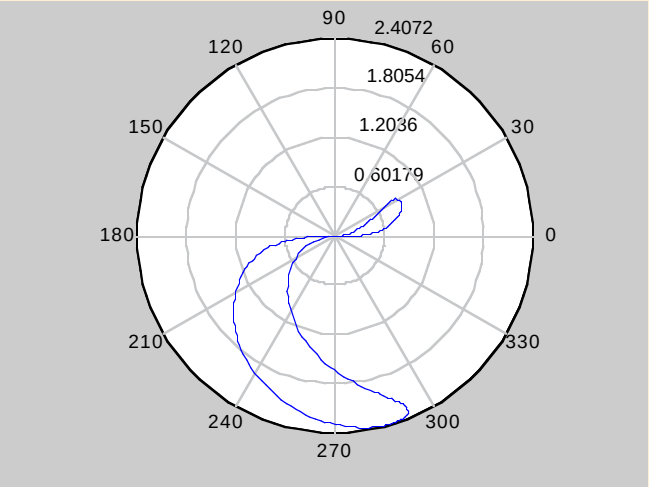
Zweidimensionale Graphik in Polarkoordinaten

Beschreibung

Der Befehl `polar(phi,r)` zeichnet die beiden Vektoren phi und r in ein polares Koordinatensystem. Die Werte des Vektors phi sind im Bogenmass [rad] angegeben. Die Werte des Vektors r entsprechen dem Radius, d.h. dem Abstand zwischen dem Ursprung und dem betreffenden Punkt der Funktion.

Beispiel

```
>> omega=[0:0.01:pi];
>> r=r=omega.*sin(2*omega);
>> phi=r.*cos(omega);
>> polar(phi,r)
```



Befehl

`plotyy`

Anwendung

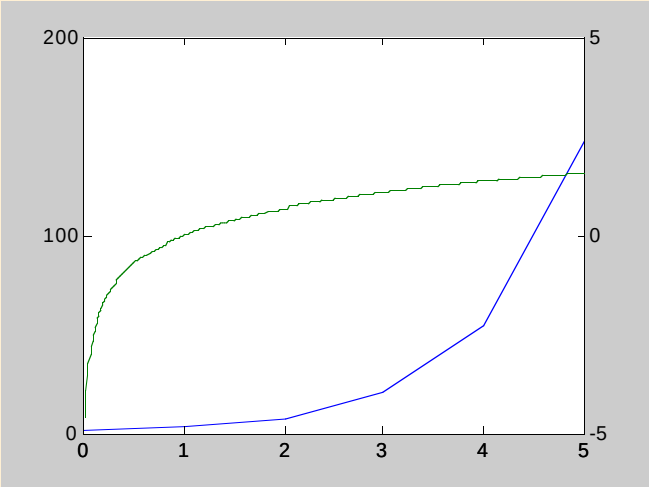
Zweidimensionale Graphik mit zwei unabhängigen, linear skalierten y-Achsen

Beschreibung

Der Befehl `plotyy(x1,y1,x2,y2)` versieht die Graphik mit zwei Ordinaten. Die linke y-Achse bezieht sich auf y_1 in Funktion von x_1 und die rechte y-Achse auf y_2 in Funktion von x_2.

Beispiel

```
>> x1=[0:1:5]; x2=[0:0.01:5];
>> y1=exp(x1); y2=log(x2);
>> plotyy(x1,y1,x2,y2)
```



3.1.2 Massstab

Befehle	Kurzbeschreibung
`axis`	Anwenderspezifische Achsendefinition [1)]
`zoom`	Vergrössern, Verkleinern
`grid`	Liniennetz
`box`	Graphik in einen Rahmen einbinden
`hold`	Im aktuellen Graphikfenster verweilen
`axes`	Position und Grösse der Graphik definieren [1)]

[1)] Beachten Sie den Unterschied von axis und `axes`!

Befehl

`axis`

Anwendung

Die Skalierung und das Erscheinungsbild der Achsen sollen definiert werden.

Beschreibung

Mit dem Befehl `axis([xmin xmax ymin ymax])` lassen sich die Grenzen der x- und der y-Achse neu definieren.

Der Befehl `axis auto` setzt für die Achsen wieder die ursprünglichen Werte ein.

Mit dem Befehl `axis equal` erhalten alle Achsen die gleiche Skalierung.

Der Befehl `axis ij` wechselt das Vorzeichen der y-Achse. Die positive y-Achse zeigt nun nach unten. `axis xy` macht `axis ij` wieder rückgängig.

Der Befehl `axis tight` passt die Achsenlänge exakt dem Bild an.

Der Befehl `axis off` schaltet alle axis-Definitionen, die “tick marks” und den Hintergrund aus. Mit `axis on` werden sie wieder aktiviert.

Siehe auch

`axis manual`, `axis fill`, `axis image`, `axis square`, `axis normal`

Befehl

`zoom`

Anwendung

Ausschnitt eines Bildes vergrössern / verkleinern.

Beschreibung

Der Befehl `zoom on` aktiviert in der aktuellen Graphik die Zoom-Funktion, er kann direkt im Command Window eingegeben werden. Mit “click and drag” wird dann im Bild ein beliebiger Ausschnitt näher herangebracht. Mit der linken Maustaste wird der Graphikausschnitt vergrössert und mit der rechten Maustaste verkleinert (bei Macintosh: “shift-click and drag”).

Der Befehl `zoom(factor)` zoomt die Achsen um den für “factor” gewählten Wert.

Der Befehl `zoom out` führt die Graphik in ihre default-Fenstergrösse zurück.

Der Befehl `zoom xon` bzw. `zoom yon` aktiviert in der aktuellen Graphik die Zoom-Funktion nur für die x- bzw. y-Achse.

Der Befehl `zoom(figurename,option)` versieht die Graphik “figurename” mit einer Zoom-Funktion. Als Option kann eine der oben genannten Zoom-Funktionen gewählt werden.

Befehl

`grid`

Anwendung

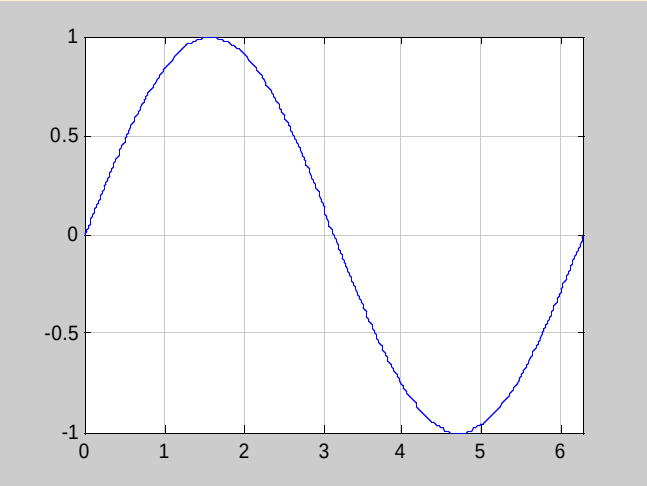
Eine Graphik mit einem Liniennetz versehen.

Beschreibung

Der Befehl `grid on` versieht die aktuelle Graphik mit einem Liniennetz. Mit dem Befehl `grid off` werden die Linien wieder deaktiviert.

Beispiel

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on
```



Siehe auch

`title`, `xlabel`, `ylabel`, `zlabel`

Befehl

`box`

Anwendung

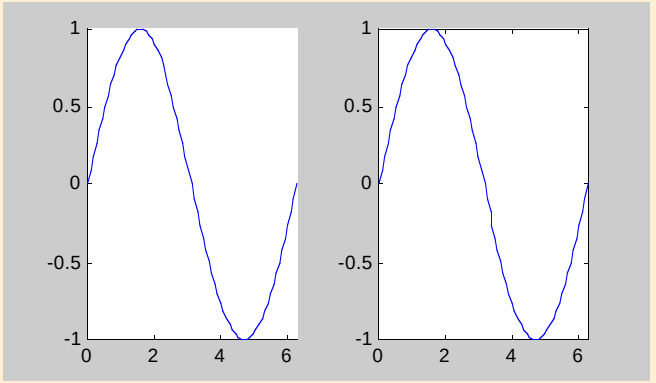
Das Graphik-Fenster mit einem Rahmen versehen.

Beschreibung

Der Befehl `box on` umrahmt das aktuelle Bild mit einem Rahmen aus dünnen, schwarzen Linien; der Befehl `box off` entfernt ihn wieder.

Beispiel

```
>> subplot(1,2,1)
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> box off
>> subplot(1,2,2)
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
```



Befehl

`hold`

Anwendung

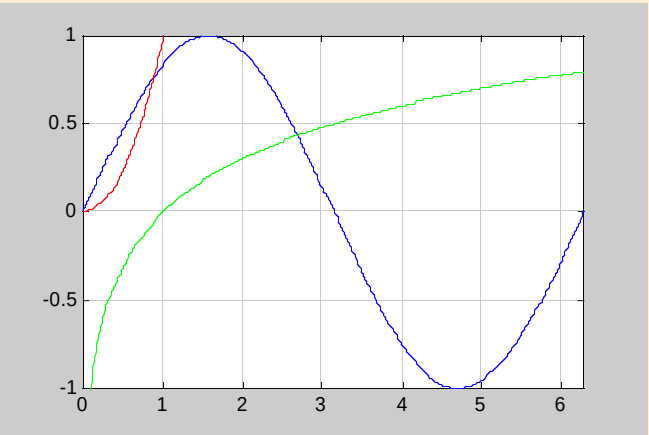
Das aktuelle Graphikfenster beibehalten

Beschreibung

Der Befehl `hold on` fixiert das aktuelle Grafikfenster, so dass weitere Funktionen im bereits bestehenden Graphikfenster positioniert werden können. Die ursprünglichen Achseinstellungen bleiben unverändert, selbst dann, wenn die neue Fuktion nicht gut in den Rahmen passt. Der Befehl `hold off` führt zum Normalbetrieb zurück, d.h. ein neuer plot-Befehl löscht die aktuelle Funktion und fügt die neue Funktion in das Fenster ein, falls nicht mit dem Befehl `figure` ein weiteres Graphikfenster geöffnet wurde.

Beispiel

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on, hold on
>> plot(x,x.^2,'r') % 'r' heisst: rote Linie”
>> plot(x,log10(x),'g') % 'g' heisst: grüne Linie
Warning: Log of zero.
```



Siehe auch

`figure`

Befehl

`axes`

Anwendung

Position und Grösse der Bildes im Graphikfenster definieren.

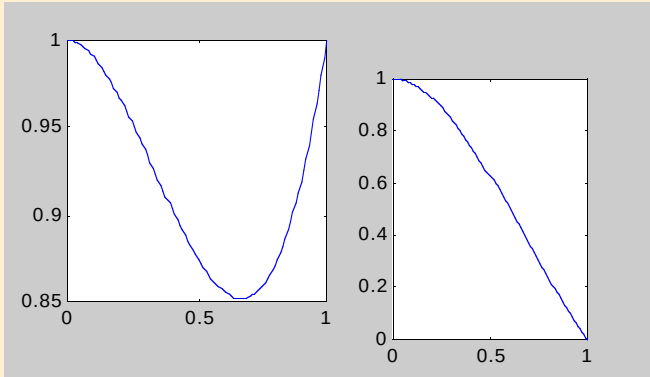
Beschreibung

Der Befehl `axes('position',[links unten Breite Höhe])` beschreibt im Graphikfenster die Position der linken unteren Ecke des Bildes und seine Abmessung. Mit Werten zwischen 0 und 1 kann nun die Position und die Grösse des Bildes festgelegt werden. Das default-Graphikfenster hat eine Breite und eine Höhe von 1.

Der Befehl `axes` generiert in einem Grafikfenster ein Koordinatensystem, in das mit `plot` ein beliebiges Bild hineingelegt werden kann. Über mehrere `axes`-Definitionen können im Fenster mehrere Bilder erzeugt werden.

Beispiel

```
>> x=[0:0.01:1];
>> axes('position',[0.05 0.15 0.4 0.8])
>> plot(x,x.^3-x.^2+1)
>> axes('position',[0.55 0.05 0.4 0.8])
>> plot(x,x.^3-2.*x.^2+1)
```



Siehe auch

`subplot`, `figure`, `gca`, `cla`

3.1.3 Beschriften von Bildern

Befehle	Kurzbeschreibung
`legend`	Bildlegende
`title`	Bildtitel
`xlabel`	Achsenbeschriftung
`text`	Text in das Bild einfügen
`gtext`	Text individuell mit der Maus einfügen

Befehl

`legend`

Anwendung

Bildlegende

Beschreibung

Der Befehl `legend('st1','st2',...)` versieht im Fenster ein Bilder mit einer Legende. Er enthält die einzelnen Strings "st_1", "st_2", usw.. Für jedes Bild in einem Graphikfenster kann ein eigener Titel gewählt werden. Der zu schreibende Text wird in Anführungszeichen `'...'` genommen.

Der Befehl `legend off` entfernt die Legende aus dem aktuellen Bild.

Der Befehl `legend('st1','st2',...,position)` positioniert die Legende mit der Angabe einer Position an einen definierten Ort in der Graphik:

0 = ist automatisch der "beste" Ort (Ort mit wenig Daten)

1 = obere rechte Ecke (Default)

2 = obere linke Ecke

3 = untere linke Ecke

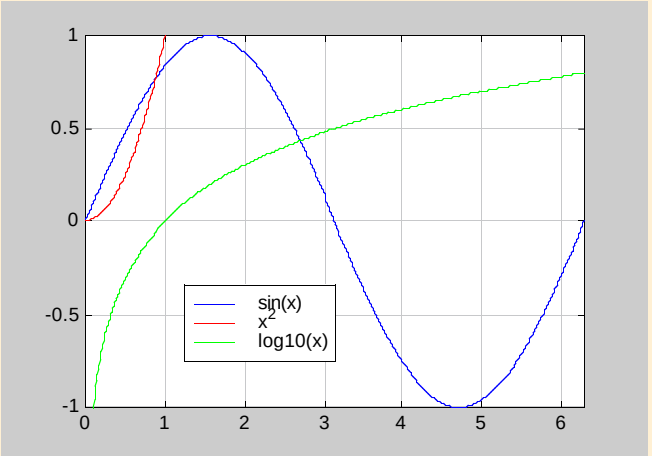
4 = untere rechte Ecke

-1 = rechts von der Graphik

Die Legende kann mit der Maus verschoben werden, indem die Legende mit der rechten Maustaste (beim Macintosh opt-click) angeklickt und an den gewünschten Ort gezogen wird.

Beispiel

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on, hold on
>> plot(x,x.^2,'r')
>> plot(x,log10(x),'g')
Warning: Log of zero.
% Text, der geschrieben werden soll, wird immer
% in Anführungszeichen '…' gebracht."0" ist Posi-
% tionsparameter der Legende.
>> legend('sin(x)','x^2','log10(x)',0)
```



Siehe auch

`plot`

Befehl

`title`

Anwendung

Bildtitel

Beschreibung

Der Befehl `title('text')` fügt einen Titel oberhalb der Graphik hinzu. Ein Text kann hoch- (“^”) und tiefgestellte (“_”), kursiv (“`it`”) geschriebene oder griechische Zeichen enthalten:

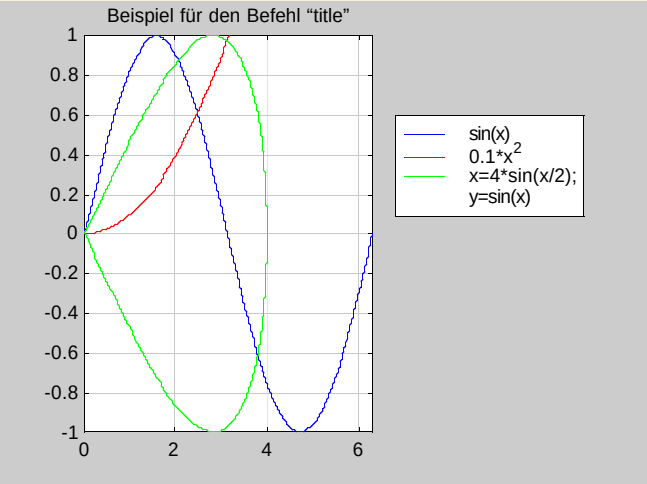
$$A_1 e^{-\alpha t} \sin \beta t \qquad (13)$$

ergibt sich aus dem String:

`'\itA_{1}e^{\alpha\itt}sin\beta\itt'`

Beispiel

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on, hold on
>> plot(x,0.1.*x.^2,'r')
>> plot(4.*sin(x./2),sin(x),'g')
>> legend('sin(x)','0.1*x^2','x=4*sin(x/2);...
   y=sin(x)',-1)
>> title('Beispiel für den Befehl “title”')
>> hold off
```



Befehl

`xlabel, ylabel`

Anwendung

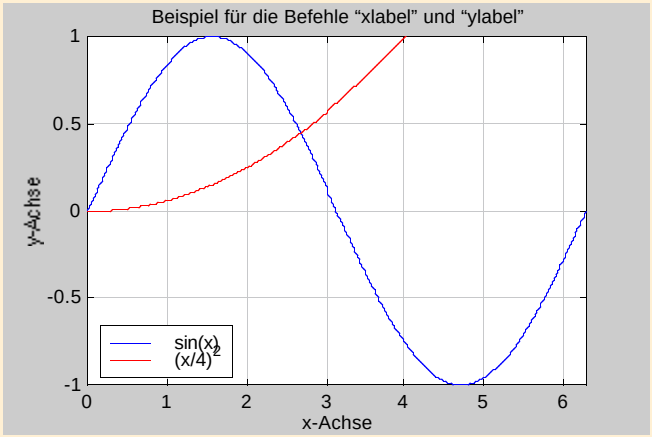
x- und y-Achse beschriften

Beschreibung

Der Befehl x`label('text')` schreibt die Zeile “text” unter die Abszisse. Der Befehl `ylabel` ist für die Beschriftung der Ordinate.

Beispiel

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on, hold on
>> plot(x,(x/4).^2,'r')
>> legend('sin(x)','(x/4)^2',3)
>> title(['Beispiel für die Befehle ',...
   '“xlabel” und “ylabel”'])
>> xlabel('x-Achse'), ylabel('y-Achse')
```



Befehl

`text`

Anwendung

Text in ein Bild einfügen

Beschreibung

Mit dem Befehl `text(x,y,'text')` kann innerhalb des Bildrahmens eine beliebige Textzeile an der Stelle (x,y) angebracht werden. x und y sind in den Koordinaten der Achsen anzugeben.

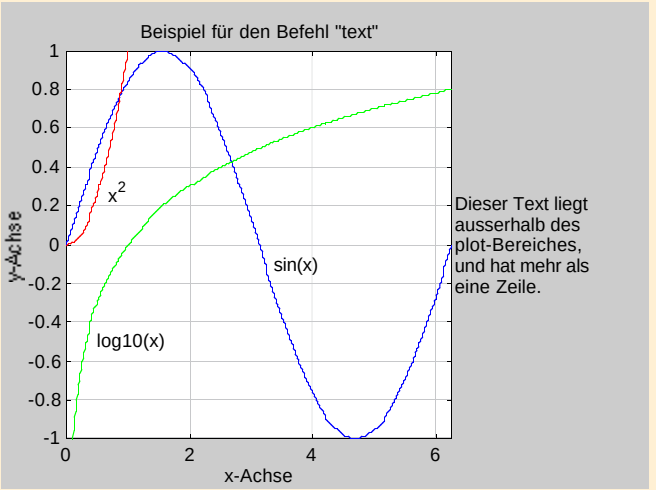
Dagegen verwendet der Befehl `text(x,y,'text','sc')` die Koordinaten des Grafikfensters, nämlich (0,0) in der unteren linken Ecke und (1,1) in der oberen rechten Ecke.

Geht ein Text über mehrere Zeilen, so kann er in eine Text-Variable geschrieben werden (siehe Beispiel).

Ein Text kann hoch- (“^”) und tiefgestellte (“_”), kursiv (“`it`”) geschriebene oder griechische Zeichen enthalten.

Beispiel

```
>> x=[0:0.01:2*pi];
>> axes('position',[0.1 0.1 0.6 0.8])
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on,hold on
>> plot(x,x.^2,'r')
>> plot(x,log10(x),'g')
Warning: Log of zero.
>> title('Beispiel für den Befehl “text”')
>> xlabel('x-Achse')
>> ylabel('y-Achse')
>> text(0.7,0.25,'x^2')
>> text(3.4,-0.1,'sin(x)')
>> text(0.5,-0.5,'log10(x)')
>> str1(1)={'Dieser Text liegt'};
>> str1(2)={'ausserhalb des'};
>> str1(3)={'plot-Bereiches,'};
>> str1(4)={'und hat mehr als'};
>> str1(5)={'eine Zeile.'};
>> text(1,0.5,str1,'sc')
```



Siehe auch

`line`, `patch`

Befehl

`gtext`

Anwendung

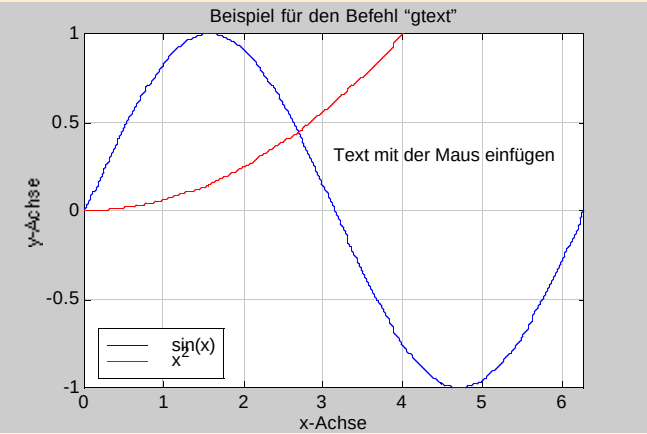
Text individuell mit der Maus einfügen

Beschreibung

Mit dem Befehl `gtext('text')` kann mit der Maus im Bild irgendein Text an beliebigem Ort eingefügt werden. Im Graphikfenster erscheint der Mauspfeil als Fadenkreuz. An der gewünschten Stelle kann durch Betätigung einer Maustaste der Text eingefügt werden.

Beispiel

```
>> x=[0:0.01:2*pi];
>> plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> grid on, hold on
>> plot(x,(x/4).^2,'r')
>> legend('sin(x)','x^2',3)
>> title('Beispiel für den Befehl “gtext”')
>> xlabel('x-Achse'), ylabel('y-Achse')
>> gtext('Text mit der Maus eingefügen')
>> hold off
```



3.1.4 Graphiken speichern oder drucken

Befehle	Kurzbeschreibung
`print`	Graphik drucken
`printopt`	Druckereinstellungen
`orient`	Papier-Orientierung beim Drucker

Befehl

`print`

Anwendung

Drucken von Graphiken

Beschreibung

Der Befehl `print` sendet eine Kopie des aktuellen Graphikfensters an den Drucker. Der Befehl `print filename` speichert eine Kopie des aktuellen Graphikfensters als PostScript-Datei im aktiven Directory unter dem Dateinamen “filename”. Mit dem Befehl `print path` kann der genaue Pfad angegeben werden, wo das aktuelle Graphikfenster als Post-Script-Datei abgelegt werden soll.

Der Befehl `print [ -ddevice ] [ -options ] <filename>` speichert die aktuelle Graphik im Format des speziell gewählten Druckertreibers und der zusätzlichen Option im aktiven Directory unter “filename” ab.

Der Befehl `help print` zeigt im Command-Window alle möglichen [-ddevice] und [-options].

Beispiel

```
>> x=[0:0.01:2*pi]; plot(x,sin(x))
>> axis([0 2*pi -1 1])
>> print % Die Graphik wird zum Drucker geschickt
>> print sinus
>> % Die Graphik wird unter “sinus” abgespeichert
>> print/home/muster/MATLAB/sinus
>> % Die Graphik wird beim Benutzer “Muster” im
>> % Subdirectory MATLAB unter “sinus” abgelegt-.
>> % Unix verwendet für den Pfad Schrägstriche
>> % “/”, Macintosh hingegen Doppelpunkte “:”.
```

Befehl

`printopt`

Anwendung

M-File mit den Druckereinstellungen

Beschreibung

Der Befehl `[pcmd,dev]=printopt` zeigt im Command-Window Druckbefehl (p(rint)c(o)m(man)d) und Treiber (dev(ice)), die der Computer verwendet. Der Druckbefehl schickt die Datei zum Drucker, und der Treiber bestimmt das Datei-Format.

pcmd-default:

Unix: lpr -s -r

Windows: COPY /B LPT1:

Macintosh: macprint

VMS: PRINT/DELETE

SGI: lp

dev-default:

Unix & VMS: -dps2

Windows: -dwin

Macintosh: -dps2

Befehl

`orient`

Anwendung

Grafikfensterorientierung beim Drucken

Beschreibung

Der Befehl `orient landscape` druckt bei print-Befehlen die Graphikfenster im Querformat.

Mit dem Befehl `orient portrait` wird das Grafikfenster im Hochformat gedruckt.

Der Befehl `orient tall` setzt das Blattformat auf Hochformat. Zusätzlich wird das Graphikfenster auf das ganze Papierblatt vergrössert bzw. verkleinert.

Der Befehl `orient` sagt im Command-Window, welche Orientierung momentan aktiv ist.

3.2 Dreidimensionale Graphik

3.2.1 Elementare dreidimensionale Graphik

Befehle	Kurzbeschreibung
`plot3`	3-D Linien-Graphik
`mesh`	3-D Graphik als Netz
`surf`	3-D Graphik als Oberfläche
`fill3`	3-D Graphik als Polygon

Befehl

`plot3`

Anwendung

Graphen im dreidimensionalen Raum

Beschreibung

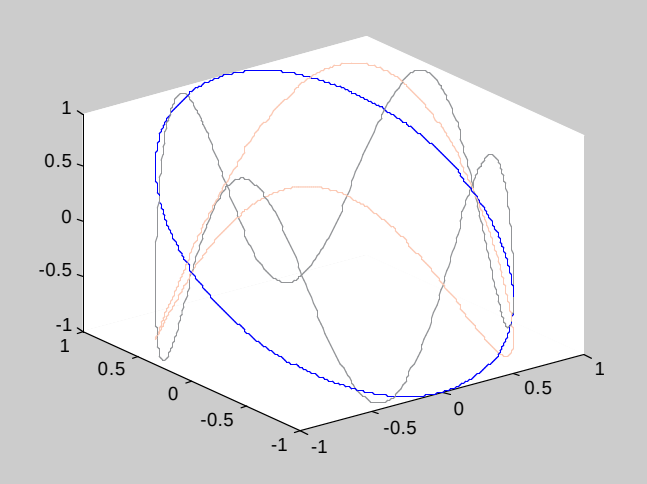
`plot3(x,y,z)` plottet im dreidimensionalen Raum die Graphen, die durch die Vektoren x, y und z gegeben sind. Die Vektoren müssen alle dieselbe Länge haben.

Bei `plot3(X,Y,Z)` zeichnet pro Kolonne einen Graphen. Die Matrizen müssen alle dieselbe Grösse haben.

Bei `plot3(x,y,z,'style')` können zusätzlich noch der Linientyp, die Plot Symbole und die Farbe des Graphen geändert werden. `help plot` listet im Command Window eine Auswahl von möglichen Liniendefinitionen auf.

Beispiel

```
>> t=[0:0.01:2*pi];
>> plot3(cos(t),sin(t),sin(t))
>> hold on
>> plot3(cos(t),sin(t),sin(2*t),'r:')
>> plot3(cos(t),sin(t),sin(4*t),'k-.')
```



Siehe auch

`plot`, `line`, `axis`, `view`

Befehl

`mesh`

Anwendung

Dreidimensionale Netz-Oberfläche

Beschreibung

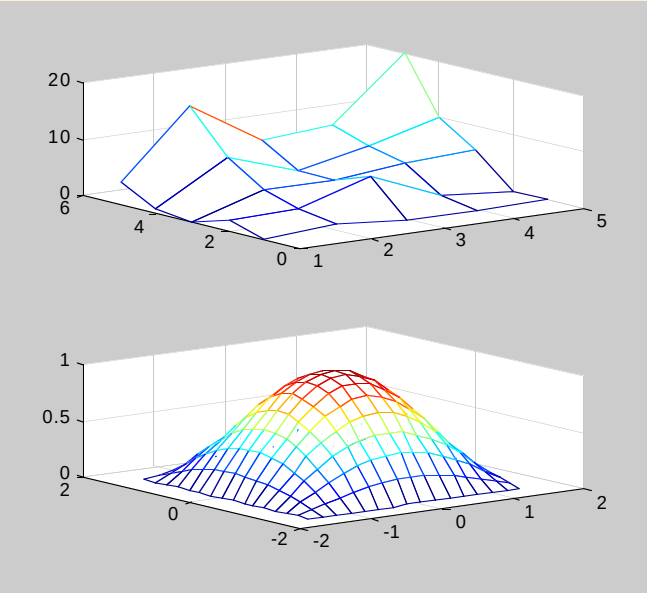
Bei `mesh(Z)` entsprechen die Werte der Matrix Z Element von $R^{n \times m}$ den z-Werten des Netzes. Für die x- und die y-Werte verwendet mesh die Kolonnen- bzw. die Zeilennummer.

Mit `mesh(Z,C)` wird das Netz zusätzlich mit Farbe versehen. Jeder Punkt erhält eine Farbe. Somit ist C ebenfalls eine n×m-Matrix.

`mesh(X,Y,Z,C)` zeichnet ein Netz in der Vogelperspektive mit Z als Funktion von X und Y. Es handelt sich hierbei um eine Funktion mit zwei Variablen. X, Y und Z sind Matrizen mit den Werten für die x-, y- und z-Koordinaten. X und Y können aber auch Vektoren der Länge m und n sein. Jeder z-Koordinate aus der Matrix Z werden dann die entsprechenden Werte des x- und y-Vektors zugewiesen. C ist ebenfalls eine Matrix und beinhaltet die Farbskala für die Graphik. Ohne C wird C = Z gesetzt.

Beispiel

```
>> Z=[0 1 0 0 0;2 2 6 1 0;0 4 4 5 6;...
   1 8 4 7 10;4 16 8 9 20];
>> x=[-pi./2:0.1:pi./2];
>> y=[-pi./2:0.1:pi./2];
>> [X,Y]=meshgrid(x,x);
>> subplot(2,1,1)
>> mesh(Z)
>> subplot(2,1,2)
>> Z=cos(Y).*cos(X);
>> mesh(x,y,Z)
```



Siehe auch

`meshgrid`, `meshc`, `meshz`, `waterfall`

Befehl

`surf`

Anwendung

Dreidimensionaler farbiger Oberflächen-Plot

Beschreibung

`surf(X,Y,Z,C)` zeichnet aus den vier Matrizen eine farbige Oberfläche. Der Aufbau der Funktion `surf` entspricht demjenigen von `mesh`. Die beiden Befehle unterscheiden sich nur in der Art der Oberfläche. `mesh` plottet ein Netz und `surf` eine geschlossene Fläche.

Bei `surf(X,Y,Z)` wurde die Farbmatrix C weggelassen. Somit ist die Farbgebung proportional zur Höhe z der Oberfläche.

Bei `surf(x,y,Z,C)` und `surf(x,y,Z)` wurden die Matrizen X und Y durch die Vektoren x und y ersetzt. Der Vektor x hat die Länge m und y die Länge n. Z ist eine n×m-Matrix. Ein Punkt in der Oberfläche hat die Koordinaten (x(j), y(i), Z(i,j)).

Der Vektor x liefert den Wert für die Kolonne von Z und y für die Zeile.

Bei surf(Z) und surf(Z,C) werden nur die Werte für die z-Koordinate definiert. Für x- und y-Koordinaten bildet MATLAB die Vektoren x = (1:m) und y = (1:n).

`surf` hat eine vordefinierte Ansicht, Farbgebung und Schattierung.

Die Ansicht (siche -> `view`) wurde in der horizontalen Ebene um -37.5 Grad gedreht und in der vertikalen um 30 Grad.

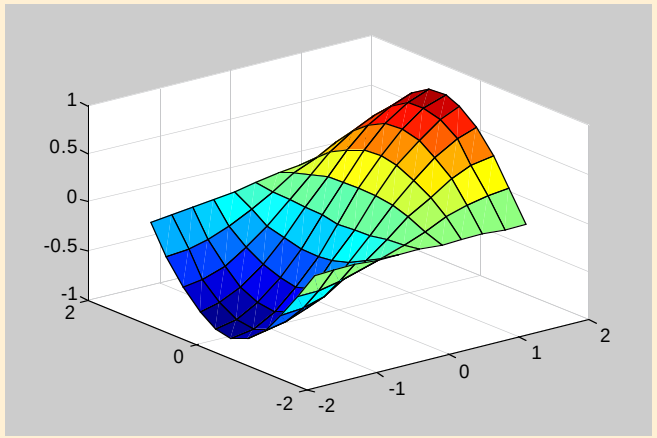
Die Graphik hat die Farbgebung `colormap('default')`. Folgende colormaps stehen zur Verfügung: `cool`, `hot`, `hsv`, `gray`, `bone`, `copper`, `pink`, `white`, `jet`, `prism`, `lines`, `colorcube`, `summer`, `autumn`, `winter`, `spring` und `flag`.

Mit `colormap([rot grün blau])` erhält die Oberfläche eine einzige Farbe. Die Skalare rot, grün und blau können Werte zwischen 0 und 1 annehmen. Z.B. [0 0 0] ist die Farbe schwarz, [1 1 1] weiss, [1 0 0] rot, [0.5 0.5 0.5] grau und [127/255 1 212/255] aquamarin. `colormap` alleine gibt eine vordefinierte Farbpalette (64×3-Matrix) mit 64 Farben wieder.

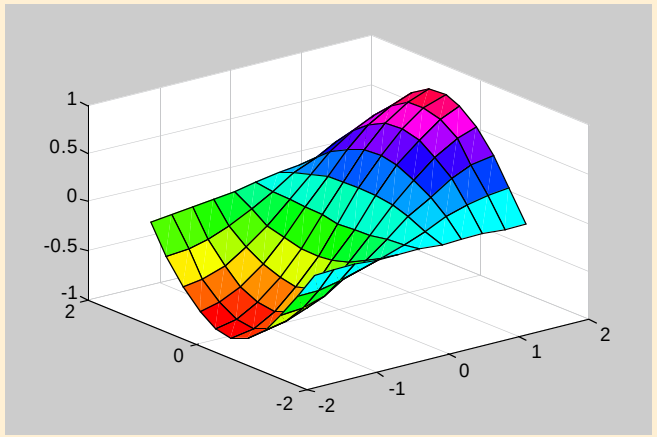
Die Graphik-Schattierung ist `shading faceted`. Es gibt noch `shading flat` und `shading interp`.

Beispiel

```
>> x=[-pi./2:0.3:pi./2];
>> y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> surf(x,y,z)
```




```
>> colormap(hsv)
```



Siehe auch

`surfc`, `surfl`, `colormap`, `view`, `shading`

Befehl

`fill3`

Anwendung

Die Graphik dreidimensional ausfüllen

Beschreibung

fill(X,Y,Z,C) plottet in den Farben der Matrix C ein dreidimensionales Polygon. Sind X, Y und Z Vektoren, so wird die Fläche unterhalb des Graphen mit Farbe ausgefüllt.

Ist C ein Skalar, so wird die Fläche monochrom. Folgende Farben sind möglich: `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, `'w'` und `'k'`.

Mit dem Vektor [rot grün blau] kann eine neue Farbe gemischt werden. Die Werte von rot, grün und blau liegen zwischen 0 und 1. Je nach Anteil rot, grün und blau ergibt sich eine neue Farbkombination.

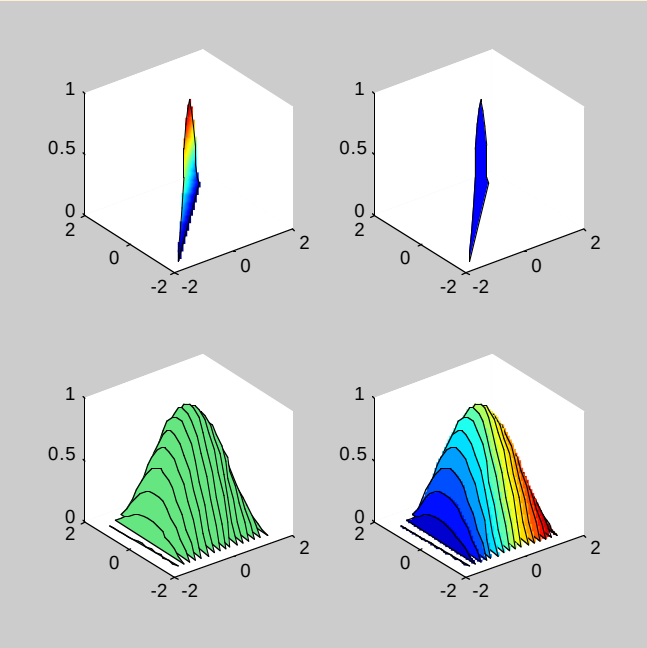
Ist C ein Vektor, so hat er die gleiche Länge wie X, Y und Z. Wird für C einer der drei Vektoren gewählt, dann ist die Farbabstufung der momentan aktive colormap proportional zur betreffenden Koordinatenachse.

Sind X, Y und Z Matrizen, so zeichnet `fill3` pro Kolonne ein Polygon und füllt es mit der entsprechenden Farbe aus. Die Farbgebung bleib gleich. Falls C ein Zeilenvektor ist, dann hat das Polygon die Schattierung `shading flat`. Für eine Matrix wird sie `shading interp`.

`shading` schattiert die Objekt-Oberfläche. `shading flat` berechnet für jede Teilfläche einer Oberfläche, die mit den Befehlen `surf`, `mesh`, `polar`, `fill` oder `fill3` gebildet wurden, die entsprechende Farbabstufung. `shading interp` interpoliert über die Farbabstufung. `shading faceted` (default-Einstellung) entspricht dem `shading flat`. Die 3-D Graphik wird jedoch zusätzlich mit schwarzen Linien versehen.

Beispiel

```
>> x=[-pi./2:0.2:pi./2];
>> y=[-pi./2:0.2:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=cos(y).*cos(x);
>> Z=cos(Y).*cos(X);
>> subplot(2,2,1)
>> fill3(x,y,z,z)
>> subplot(2,2,2)
>> fill3(x,y,z,'b')
>> subplot(2,2,3)
>> fill3(X,Y,Z,[0.4 0.9 0.5])
>> subplot(2,2,4)
>> fill3(X,Y,Z,X)
```



Siehe auch

`patch`, `fill`, `colormap`, `shading`

3.2.2 Projektionsarten einer Graphik

Befehle	Kurzbeschreibung
`view`	Ansicht definieren
`viewmtx`	Blickpunkt und Blickrichtung definieren
`rotate3d`	3-D Graphik rotieren

Befehl

`view`

Anwendung

Bei einer 3-D Graphik den Blickwinkel spezifizieren

Beschreibung

Mit `view(az,el)` kann in einem dreidimensionalen Plot der Blickwinkel beliebig eingestellt werden, bzw. die Graphik-Box ist um zwei Achsen drehbar. az steht für azimuth und definiert die horizontale Rotation in Grad. Für einen positiven Winkel dreht sich die Graphik entgegen dem Uhrzeigersinn um die z-Achse. el beschreibt die Anheben (elevation) bzw. Senken der Graphik in Grad. Bei einem positiven Winkel befindet sich der Betrachter in der Vogelperspektive, bei negativem Winkel in der Froschperspektive.

`view([x y z])` setzt den Blickwinkel in kartesischen Koordinaten.

`view(2)` stellt für die 2-D Ansicht den vordefinierten Blickwinkel `view(0,90)` ein.

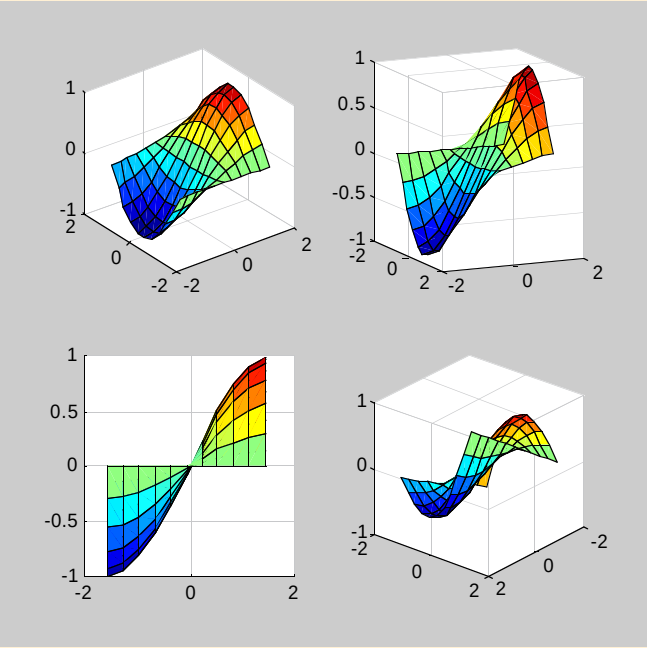
`view(3)` stellt für die 3-D Ansicht den vordefinierten Blickwinkel `view(-37.5,30)` ein.

`T=view` speichert die view der aktuellen Graphik in der Variable T als 4×4-Matrix ab.

`view(T)` weist einer aktuellen Graphik die in der Variable T gespeicherte view zu.

Beispiel

```
>> x=[-pi./2:0.3:pi./2]; y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> subplot(2,2,1)
>> surf(x,y,z)
>> view(3)
>> subplot(2,2,2)
>> surf(x,y,z)
>> view(25,-10)
>> subplot(2,2,3)
>> surf(x,y,z)
>> view(0,0)
>> subplot(2,2,4)
>> surf(x,y,z)
>> view(-40,-25)
>> colormap(jet)
```



Befehl

`viewmtx`

Anwendung

Transformationsmatrix für den Blickwinkel generieren.

Beschreibung

`T=viewmtx(az,el)` weist wie bei `T=view(az,el)` der Variable T die 4×4-Transformationsmatrix zu. Die Ansicht der aktuellen Graphik wird dabei nicht verändert.

Mit `T=viewmtx(az,el,phi)` wird die Graphik durch ein Objektiv betrachtet. Der Linsenwinkel wird in Grad angegeben. phi = 0 Grad definiert die orthogonale Projektion.10 Grad entspricht einem Teleobjektiv, 25 Grad einem Normalobjektiv und 60 Grad einem Weitwinkelobjektiv.

Bei `T=viewmtx(az,el,phi,tp)` wird mit tp=[xp,yp,zp] einen Fluchtpunkt gestetzt.

Beispiel

```
>> T1=viewmtx(-20,10)
T1 =
       0.9397   -0.3420         0         0
       0.0594    0.1632    0.9848         0
      -0.3368   -0.9254    0.1736         0
            0         0         0    1.0000
>> T2=viewmtx(-20,10,10)
T2 =
       0.9397   -0.3420         0   -0.2988
       0.0594    0.1632    0.9848   -0.6037
      -0.3368   -0.9254    0.1736   -0.3217
       0.0417    0.1145   -0.0215    1.0398
>> T3=viewmtx(-20,10,10,[0 0 1])
T3 =
       0.9397   -0.3420         0         0
       0.0594    0.1632    0.9848   -0.9848
      -0.3368   -0.9254    0.1736   -0.1736
       0.0417    0.1145   -0.0215    1.0215
```

Befehl

`rotate3d`

Anwendung

Interaktives Rotieren einer dreidimensionalen Graphik.

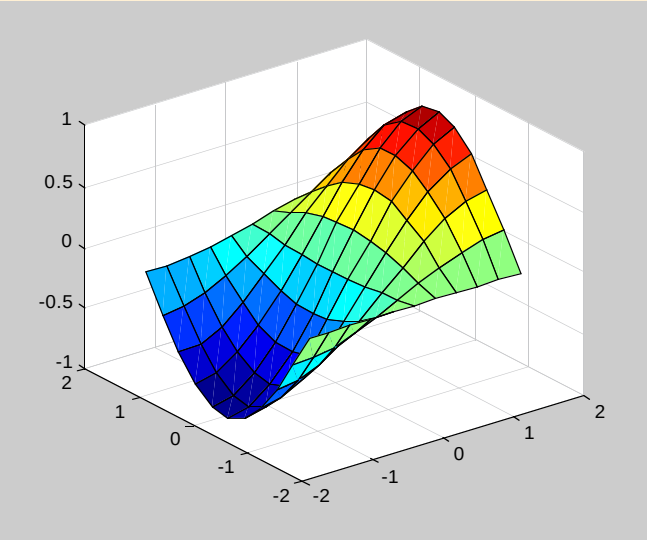
Beschreibung

`rotate3d` on aktiviert in der aktuellen Graphik die Maus gesteuerte 3-D Rotation. Die Graphik-Ansicht kann damit beliebig verändert werden.

Mit `rotate3d off` wird sie wieder deaktiviert.

Beispiel

```
>> x=[-pi./2:0.3:pi./2];
>> y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> surf(x,y,z)
>> colormap(jet)
>> rotate3d on
```



Siehe auch

`zoom`

3.2.3 Dreidimensionale Graphik beschriften

Befehle	Kurzbeschreibung
`zlabel`	z-Achse
`colorbar`	Farbskala

Befehl

`zlabel`

Anwendung

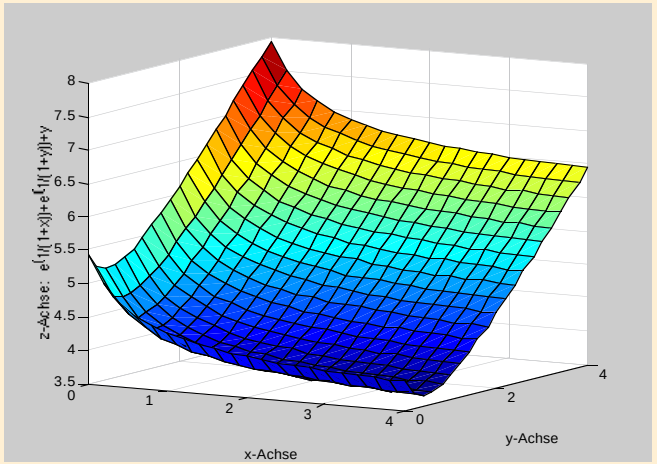
Beschriftung der z-Achse

Beschreibung

`zlabel('text')` versieht die z-Achse mit der Aufschrift “text”.

Beispiel

```
>> x=[0:0.2:4]; y=[0:0.2:4];
>> [X,Y]=meshgrid(x,y);
>> surf(X,Y,exp(1./(1+X))+exp(1./(1+Y))+Y)
>> view(30,10)
>> xlabel('x-Achse');ylabel('y-Achse')
>> zlabel('z-Achse:   e^(1/(1+x))+e^(1/(1+y))+y')
```



Siehe auch

`xlabel`, `ylabel`, `title`, `text`

Befehl

`colorbar`

Anwendung

An den aktuellen Plot angepasste Farbskala

Beschreibung

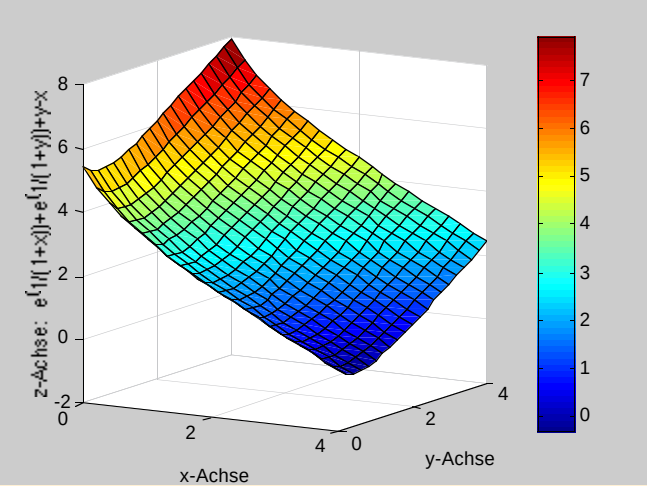
Mit `colorbar('vert')` erscheint in der aktuellen Graphik eine vertikale Farbskala. In einem 3-D Plot bezieht sie sich auf die Werte der z-Achse.

`colorbar('horiz')` zeichnet eine vertikale Farbskala. Im 3-D Plot ist die Farbgebung ebenfalls auf die z-Achse abgestimmt.

`colorbar` alleine fügt der Graphik entweder eine vertikale Farbskala hinzu, oder die bestehende Farbskala wird aktualisiert.

Beispiel

```
>> x=[0:0.2:4];y=[0:0.2:4];
>> [X,Y]=meshgrid(x,y);
>> surf(X,Y,exp(1./(1+X))+exp(1./(1+Y))+Y-X)
>> view(30,10)
>> xlabel('x-Achse');ylabel('y-Achse')
>> zlabel('z-Achse:  e^(1/(1+x))+e^(1/(1+y))+y-x)
>> colorbar
```



3.3 Spezielle Graphen

Befehle	Kurzbeschreibung
`fill`	2-D Polygon ausfüllen
`fplot`	Plotten einer Funktion
`hist`	Histogramm
`pie`	Kuchen-Diagramm
`stem`	Mengen-Graphik
`stairs`	Treppenfunktion
`contour`	Konturplot

Befehl

`fill`

Anwendung

Zweidimensionale Kurve einer Funktion y = f(x) mit einer beliebigen Farbe ausfüllen.

Beschreibung

`fill(x,y,c)` füllt diejenige Fläche mit der Farbe c aus, welche von der Geraden, die den Endpunkt von f(x) mit dem Anfang verbindet, und der Funktion y = f(x) selbst umgeben wird.

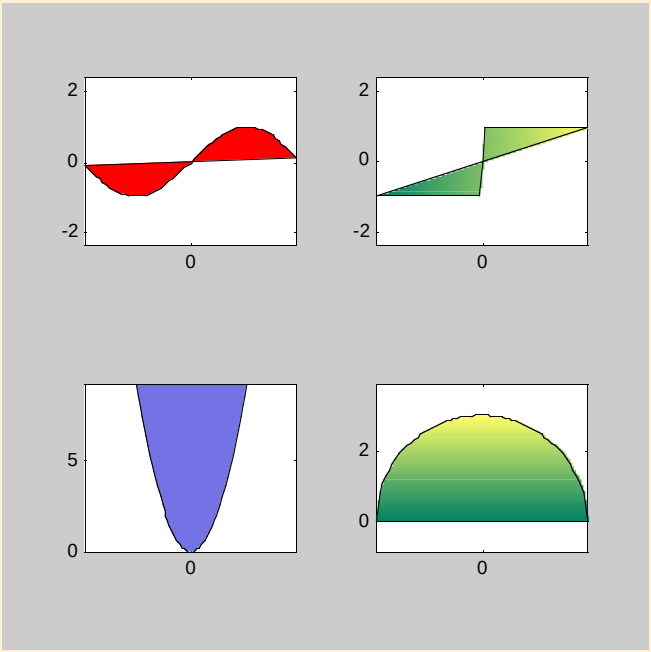
Falls für c `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, `'w'`, `'k'` oder der Farbvektor [rot grün blau] eingesetzt wird (siehe -> `fill3`), plottet MATLAB eine monochrome Fläche.

Ist c eine Vektor derselben Länge wie x und y, dann verwendet MATLAB entweder die im Vektor c definierten Farben, oder für c = x bzw. c = y die Farbpalette der aktuellen `colormap` (siehe -> `surf`).

Sind in `fill(X,Y,C)` X und Y Matrizen derselben Grösse, so wird pro Kolonne ein Polygon gezeichnet. C kann ein Vektor aber auch eine Matrix sein. Beim Vektor ist die Schattierung der Fläche `shading flat`, bei der Matrix `shading interp`.

Beispiel

```
>> x=[-3:0.1:3];
>> y1=sin(x);
>> y2=sign(x);
>> y3=x.^2;
>> y4=sqrt(9-x.^2);
>> colormap(summer)
>> subplot(2,2,1)
>> fill(x,y1,'r')
>> subplot(2,2,2)
>> fill(x,y2,x)
>> subplot(2,2,3)
>> fill(x,y3,[0.45 0.2 0.9])
>> subplot(2,2,4)
>> fill(x,y4,y4)
```



Siehe auch

`patch`, `fill3`, `colormap`, `shading`

Befehl

`fplot`

Anwendung

Funktion mit einer Variable plotten.

Beschreibung

`fplot('f',lim)` zeichnet eine beliebige Funktion f = f(x) im Bereich lim = [x_{min} x_{max}]. Mit lim = [x_{min} x_{max} y_{min} y_{max}] werden zusätzliche Schranken für die y-Achse gesetzt.

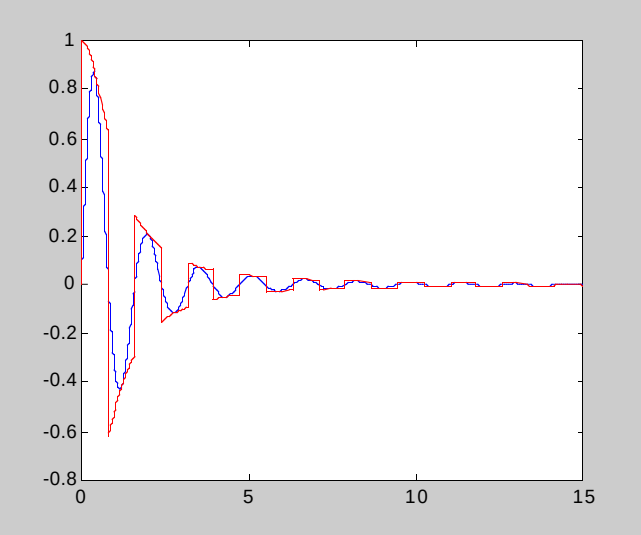
`fplot('f', lim, tol)` mit tol<1 definiert die Toleranz des relativen Fehlers. Die voreingestellte Toleranz ist 2e-3 bzw. 0.2%.

`fplot('f',lim,N)` berechnet zwischen x_{min} und x_{max} für die Funktion f = f(x) N+1 Punkte.

`fplot('f',lim,'LineSpec')` definiert mit LineSpec den Linien-Typ der Funktion. Alle möglichen “line specifications” werden mit `help plot` aufgelistet.

Beispiel

```
>> fplot('sin(4*x)*(1/(1+x^2))',[0 15],1e-4)
>> fplot('sign(sin(4*x))*(1/(1+x^2))',...
   [0 15],1e-4)
```



Befehl

`hist`

Anwendung

Aus einem Vektor mit Daten ein Histogramm erzeugen.

Beschreibung

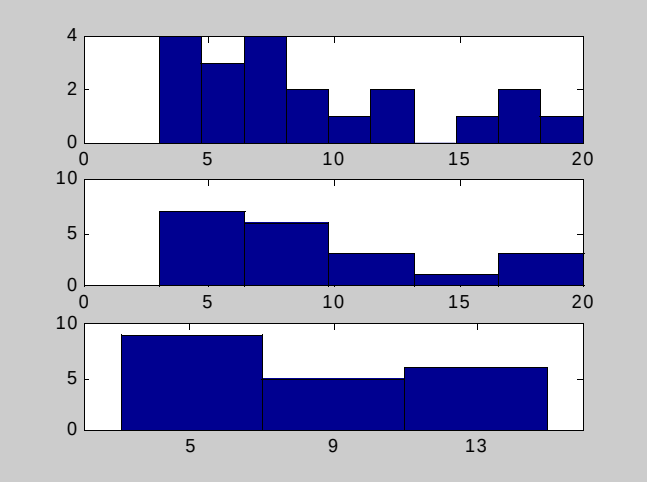
`hist(x)` zeichnet ein Histogramm mit den in x gespeicherten Daten. Es ist in 10 gleichmässig verteilte Intervalle unterteilt. Pro Intervall gibt es die Anzahl Elemente an, die es enthält. Wenn x eine Matrix ist, plottet `hist` pro Kolonne ein Histogramm.

`hist(x,n)` hat n Intervalle.

`hist(x,y)` berechnet die Verteilung von x bezüglich y. y ist ein Vektor, deren Elemente in aufsteigender Ordnung aufgelistet sind. Jedes einzelne Element von y entspricht einem Zentrum.

Beispiel

```
>> x=[9 3 20 4 6 12 15 3 8 5 11 17 7 4 9 6 18 13
7 8];
>> y=[5 9 13];
>> subplot(3,1,1),hist(x)
>> subplot(3,1,2),hist(x,5)
>> subplot(3,1,3),hist(x,y)
```



Befehl

`pie`

Anwendung

Aus Daten Kuchendiagramm erstellen.

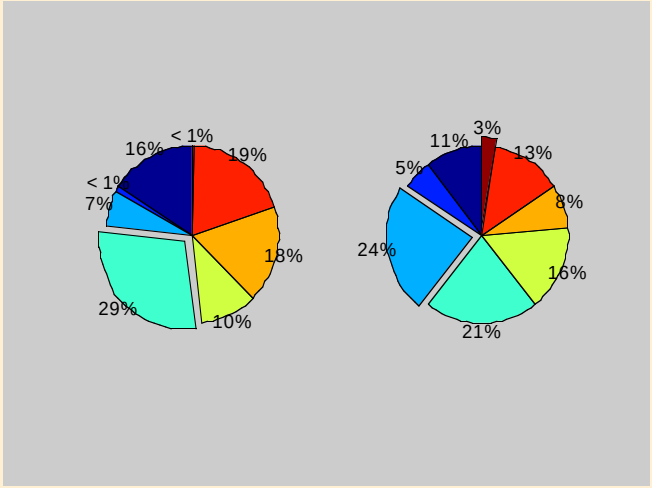
Beschreibung

`pie(x)` stellt die Daten aus dem Vektor x in einem Kuchendiagramm dar. Die Elemente von x werden mit der Summe der x-Werte dividiert. Damit ist die Grösse von jedem einzelnen Kuchenstück in % gegeben.

`pie(x,explode)` zieht mit “explode” die gewünschte Stücke aus dem Kuchen. explode ist ein Vektor derselben Länge wie x. Seine Elemente haben entweder den Betrag 0, d.h. das entsprechende Stück von x verbleibt im Kuchen, oder den Betrag 1, d.h. das dazugehörende Stück von x wird aus dem Kuchen herausgezogen.

Beispiel

```
>> x1=[3.6 0.2 1.5 6.5 2.3 4.1 4.4 0.1];
>> x2=[4 2 9 8 6 3 5 1];
>> subplot(1,2,1)
>> pie(x1,[0 0 0 1 0 0 0 0])
>> subplot(1,2,2)
>> pie(x2,(x2==min(x2))+(x2==max(x2)))
```



Befehl

`stem`

Anwendung

Gespeicherte Daten als Kreise mit Linien plotten.

Beschreibung

`stem(y)` zeichnet eine Verteilung der Daten aus dem Vektor y. Jeder Wert aus y ist im Plot mit einem Kreis und einer Linie versehen. Auf der Abszisse wird jedes Element aus x fortlaufend eingereiht. Auf der Ordinate kann sein Wert abgelesen werden. Der entsprechende Wert ist mit einem Kreis gekennzeichnet.

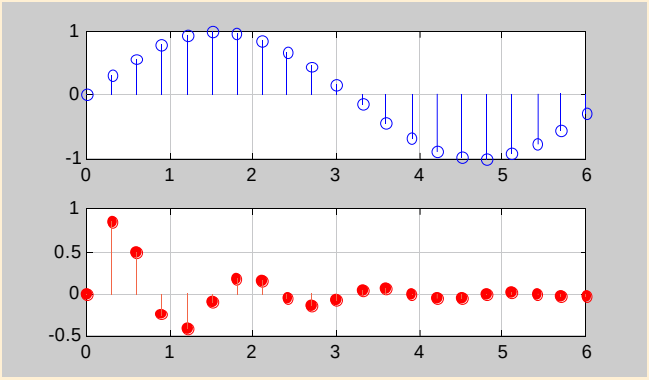
Bei `stem(x,y)` entsprechen die Elemente aus dem x-Vektor den x-Werten und diejenigen aus dem y-Vektor den y-Werten.

`stem(...,'filled')` malt den Kreis mit der entsprechenden Farbe aus.

Mit `stem(...,'linespec')` kann der Linien-Typ gewählt werden. Alle möglichen “line specifications” werden mit `help plot` aufgelistet.

Beispiel

```
>> x=[0:0.3:6];
>> subplot(2,1,1),stem(x,sin(x)), grid on
>> subplot(2,1,2), stem(x,sin(4.*x).*(1./...
   (1+x.^2)),'filled','r--'), grid on
```



Siehe auch

`plot`, `bar`

Befehl

`stairs`

Anwendung

Gespeicherte Daten mit einer Treppenfunktion darstellen.

Beschreibung

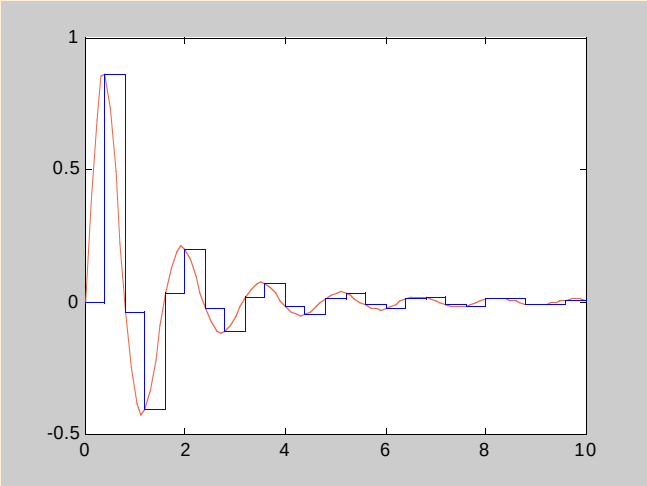
`stairs(y)` zeichnet mit den Werten aus dem Vektor y eine Treppenfunktion. Ohne x-Vektor macht MATLAB pro Element ein Schritt von Betrag Eins.

Bei `stairs(x,y)` entsprechen die Elemente aus dem x-Vektor den x-Werten und diejenigen aus dem y-Vektor den y-Werten.

Mit `stairs(...,'style')` kann der Linien-Stil gewählt werden. Alle möglichen “styles” werden mit `help plot` aufgelistet.

Beispiel

```
>> x1=[0:0.1:10];y1=sin(4.*x1).*(1./(1+x1.^2));
>> plot(x1,y1,'r--'), hold on
>> x2=[0:0.4:10];y2=sin(4.*x2).*(1./(1+x2.^2));
>> stairs(x2,y2), hold off
```



Siehe auch

`bar`

Befehl

`contour`

Anwendung

Höhenlinien von einer dreidimensionalen Graphik plotten.

Beschreibung

`contour(Z)` zeichnet einen Konturplot mit den Werten aus der Matrix Z. Die Elemente der Matrix Z entsprechen den Werten auf der z-Achse. Die Kolonnenzahlen ergeben die Koordinaten auf der x-Achse und die Zeilenzahlen die Werte auf der y-Achse. Die Höhen für die einzelnen Höhenlinien werden automatisch ausgewählt.

Mit `contour(x,y,Z)` werden die x- und y-Koordinaten explizit mitgeliefert.

`contour(Z,N)` und `contour(x,y,Z,N)` verwendet für den Konturplot N Höhenlinien.

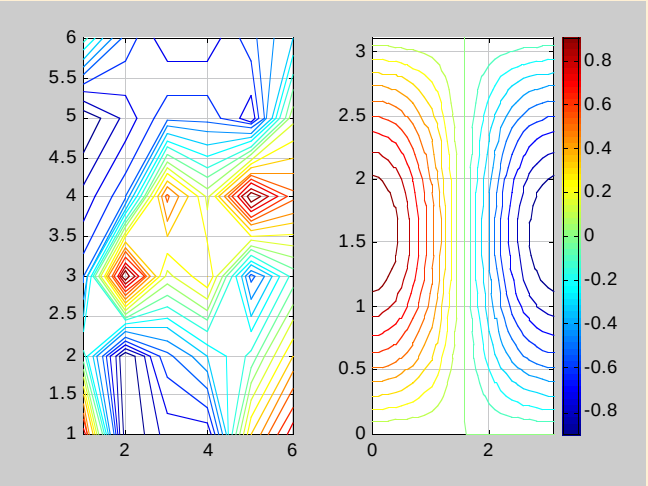
`contour(Z,v)` und `contour(x,y,Z,v)` zeichnet all jene Höhenlinien, deren Höhen im Vektor v angegeben werden.

Mit `contour(Z,[v v])` plottet MATLAB eine einzige Höhenlinie bei der Höhe v.

Mit `contour(...,'linespec')` kann der Linien-Typ gewählt werden. Alle möglichen “line specifications” werden mit `help plot` aufgelistet.

Beispiel

```
>> x=[0:0.1:pi];
>> y=[0:0.1:pi];
>> v=[-1:0.1:1];
>> [X,Y]=meshgrid(x,y);
>> Z=[8 0 1 1 6 8;
      4 0 2 3 4 6;
      2 9 5 6 2 4;
      1 2 7 5 9 7;
      0 1 2 2 1 5;
      4 2 1 1 2 3];
>> subplot(1,2,1)
>> contour(Z,20)
>> subplot(1,2,2)
>> contour(x,y,cos(X).*sin(Y),v)
>> colorbar
```



Siehe auch

`contour3`, `contourf`, `clabel`, `colorbar`

3.4 Graphik-Handhabung

3.4.1 Das Graphik-Fenster

Befehle	Kurzbeschreibung
`figure`	Graphik-Fenster
`gcf`	Zugriff auf das aktuelle Graphik-Fenster
`clf`	Graphik im aktuellen Fenster löschen
`close`	Graphik-Fenster schliessen
`refresh`	Graphik-Fenster erneuern

Befehl

`figure`

Anwendung

Ein Fenster für eine neue Graphik öffnen

Beschreibung

`figure` alleine öffnet ein neues Graphik-Fenster, das direkt zum aktiven Fenster wird.

Der Befehl `figure(N)` wechselt zum Fenster mit der Nr. N, auf das nun aktiv zugegriffen werden kann. N ist eine positive ganze Zahl. Falls `figure(N)` noch nicht existiert, wird eine neues Fenster mit der Nr. N geöffnet.

`figure(gcf)` zeigt das aktive Graphik-Fenster an, indem es am Bildschirm in den Vordergrund rückt.

Die Befehle `get(N)` und `set(N)` liefern eine Liste der Fenster-Einstellungen und deren aktuelle Werte.

Siehe auch

`subplot`, `axes`

Befehl

`gcf`

Anwendung

Zugriff auf die aktuelle Graphik haben.

Beschreibung

`gcf` alleine zeigt an, welches Graphik-Fenster das aktive ist. Nur in das aktive Fenster kann mit den Befehlen wie `plot`, `titel`, `surf`, `mesh`, `bar`, usw. gezeichnet werden. Ein Fenster wird mit dem Befehl `figure(N)` aktiviert.

Der Befehl `get(gcf)` liefert vom aktiven Fenster die Liste der Fenster-Einstellungen und deren aktuelle Werte. Mit dem Befehl `set` können sie geändert werden.

Siehe auch

`gca`, `gcbo`, `gco`, `gcbf`

Befehl

`clf`

Anwendung

Das Graphik-Fenster säubern, indem das Gezeichnete gelöscht wird.

Beschreibung

clf löscht alle Graphen und Achsen im aktiven Fenster. Mit den Befehlen wie `plot`, `titel`, `surf`, `mesh`, `bar`, usw. können nun von neuem Plots in das gesäuberte Fenster eingefügt werden.

`clf reset` löscht alle Graphen und Achsen im aktiven Fenster. Zusätzlich werden alle Fenster-Einstellungen, falls sie verändert wurden, in die voreingestellten Werte zurückgesetzt. Die Fenster-Position und die Einheiten sind davon ausgenommen.

Siehe auch

`cla`, `reset`, `hold`

Befehl
`close`

Anwendung
Das aktuelle Fenster schliessen.

Beschreibung
`close` alleine schliesst das aktive Fenster.

Mit `close(N)` wird das Fenster mit der Nr. N geschlossen.

`close('name')` schliesst das Fenster mit dem entsprechenden Namen.

`close all` schliesst alle offenen Graphik-Fenster.

Siehe auch
`delete`

Befehl
`refresh`

Anwendung
Die Graphik im aktiven Fenster wieder auffrischen.

Beschreibung
Mit `refresh` alleine wird die Graphik im aktiven Fenster erneuert, d.h. sie wird noch einmal gezeichnet.

Mit `refresh(fig)` wird das Fenster “fig” noch einmal gezeichnet.

3.4.2 Achsenkontrolle

Befehle	Kurzbeschreibung
`gca`	Achsen-Angabe der aktiven Graphik
`cla`	Die aktiven Achsen löschen
`caxis`	Das Intervall der Farbskala festlegen
`ishold`	Nach dem hold-Status fragen

Befehl
`gca`

Anwendung
Die Numerierung der aktiven Achsen anzeigen.

Beschreibung
`gca` gibt mit einem numerischen Wert die Numerierung der Achsen im aktiven Fenster an. Bei der Verwendung eines Plot-Befehls wie `plot`, `title`, `mesh`, `surf`, `fill3`, usw. erhalten die Achsen im Plot eine bestimmte Nummer.

Mit den Befehlen `axes` oder `subplot` wird die Achsen-Numerierung der aktiven Achsen geändert, oder es werden neue Achsen mit einer neuen Nummer geschaffen.

Siehe auch
`axes`, `subplot`, `delete`, `hold`, `gcf`

Befehl
`cla`

Anwendung
Alles innerhalb der Achsen löschen.

Beschreibung
`cla` löscht alles innerhalb der Plot-Region: Linien, Text, Patches, Oberflächen und Bilder.

`cla reset` löscht wie `cla` alles innerhalb der Achsen, und stellt zusätzlich alle Achsen-Einstellungen auf die vordefinierten Werte zurück. Die Position der Achsen ist davon ausgenommen.

Siehe auch
`reset`, `hold`

Befehl

`caxis`

Anwendung

Das Intervall der Farbskala definieren.

Beschreibung

`caxis(v)` setzt mit dem 1×2 Vektor v die Farbskala für die `surface`- und `patch`-Objekte. Sie werden mit den Befehlen `mesh`, `pcolor` und `surf` erzeugt. cmin und cmax aus dem Vektor v = [c_{min} c_{max}] stehen für die erste und letzte Farbe in der aktuellen Farbskala.

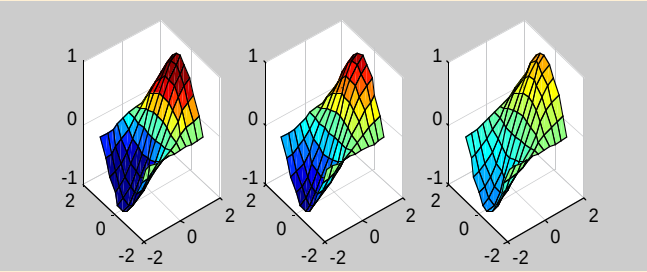
`caxis('manual')` fixiert die Farbskala beim aktuellen Intervall.

`caxis('auto')` stellt die Farbskala auf den vordefinierten Wert zurück.

`caxis` allein gibt das aktuelle Intervall an.

Beispiel

```
>> x=[-pi./2:0.3:pi./2];
>> y=[-pi./2:0.3:pi./2];
>> [X,Y]=meshgrid(x,y);
>> z=[cos(Y).*sin(X)];
>> subplot(1,3,1), surf(x,y,z), caxis([-0.6 0.6])
>> subplot(1,3,2), surf(x,y,z), caxis('auto')
>> subplot(1,3,3), surf(x,y,z), caxis([-2 2])
```



Siehe auch

`colormap`, `axes`, `axis`

Befehl

`ishold`

Anwendung

Den hold-Status anzeigen

Beschreibung

`ishold` gibt Auskunft über den Status vom `hold`. Für 1 gilt `hold on` und für 0 `hold off`.

Bei `hold on` werden der aktuelle Plot und alle Achsen-Einstellungen “eingefroren”, damit der aktiven Graphik weitere Graphen oder der Graphik untergeordnete Befehle hinzugefügt werden können.

Siehe auch

`hold`, `newplot`, `figure`, `axes`

3.4.3 Linien, Flächen und Belichtung

Befehle	Kurzbeschreibung
`line`	Eine Linie/Gerade plotten
`patch`	Ein Feld erzeugen
`surface`	Eine Oberfläche generieren
`light`	Lichtquelle definieren

Befehl

`line`

Anwendung

Eine Linie bzw. Gerade der Graphik hinzufügen.

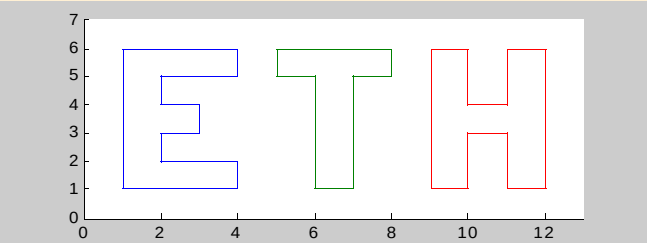
Beschreibung

`line(X,Y)` zeichnet in der aktiven Graphik Linien aus den Kolonnenvektoren X und Y. Sind X und Y ein Skalar, so erscheint an der Stelle (x,y) ein Punkt. Sind X und Y Matrizen, so plottet MATLAB pro Kolonne eine Linie.

`line(X,Y,Z)` erzeugt eine Linie im 3-D Raum.

Beispiel

```
>> ETHx=[1 4 4 2 2 3 3 2 2 4 4 1 1;...
   6 7 7 7 8 8 7 6 5 5 6 6 6;...
   9 10 10 11 11 12 12 11 11 10 10 9 9];
>> ETHy=[1 1 2 2 3 3 4 4 5 5 6 6 1;...
   1 1 3 5 5 6 6 6 6 5 5 3 1;...
   1 1 3 3 1 1 6 6 4 4 6 6 1];
>> line(ETHx',ETHy'), axis([0 13 0 7])
```



Siehe auch

`text`, `plot`, `plot3`

Befehl

`patch`

Anwendung

Einen “patch” bzw. eine Fläche der Graphik anbringen

Beschreibung

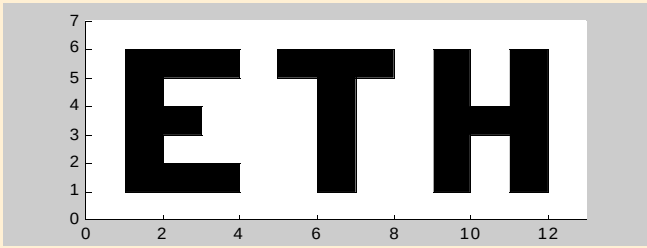
`patch(X,Y,C)` fügt in der aktiven Graphik eine Fläche in beliebiger Form hinzu. Patches werden für die Darstellung von Objekten wie Flugzeuge oder Autos verwendet. Die Daten für den “patch” befinden sich in den beiden Kolonnenvektoren X und Y. Sind X und Y Matrizen, so wird pro Kolonne eine Fläche gezeichnet. C definiert die Farbe der Fläche. Falls für C `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, `'w'`, `'k'` oder der Farbvektor [rot grün blau] eingesetzt wird (siehe -> `fill3`), plottet MATLAB eine monochrome Fläche.

Ist C eine Vektor derselben Länge wie X und Y, dann verwendet MATLAB entweder die im Vektor C definierten Farben, oder für C = X bzw. C = Y die Farbpalette der aktuelle `colormap` (siehe -> `surf`).

`patch(X,Y,Z,C)` zeichnet eine Fläche im 3-D Raum.

Beispiel

```
>> ETHx=[1 4 4 2 2 3 3 2 2 4 4 1;...
   6 7 7 7 8 8 7 6 5 5 6 6;...
   9 10 10 11 11 12 12 11 11 10 10 9];
>> ETHy=[1 1 2 2 3 3 4 4 5 5 6 6;...
   1 1 3 5 5 6 6 6 6 5 5 3;...
   1 1 3 3 1 1 6 6 4 4 6 6];
>> patch(ETHx',ETHy','k'), axis([0 13 0 7])
```



Siehe auch

`fill`, `fill3`, `text`, `shading`

Befehl

`surface`

Anwendung

Der Graphik eine Oberfläche hinzufügen.

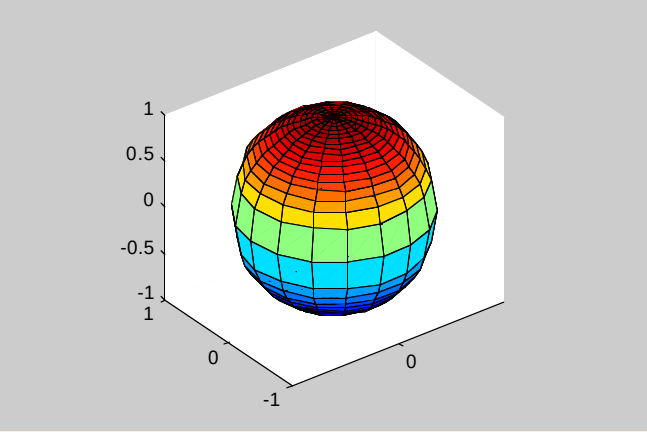
Beschreibung

`surface(X,Y,Z,C)` generiert in der aktiven Graphik mit den Vektoren oder Matrizen X und Y und der Matrix Z eine Oberfläche. Sie ist in kleine rechteckige Flächen unterteilt. `surface` eignet sich für die Darstellung einer Topographie oder eines Kennfeldes. Der Vektor oder die Matrix C beschreibt die Farbgebung der Oberfläche.

Bei `surface(X,Y,Z)` wird C gleich Z gesetzt, d.h. die Farbabstufung ist proportional zur Oberflächenhöhe.

Beispiel

```
>> r1=0:1/18:1;r2=ones(1,18)-r1(2:19);r=[r1 r2];
>> t=0:pi/9:4*pi;
>> [R,T]=meshgrid(r,t);
>> X=abs(R).*cos(T);Y=abs(R).*sin(T);
>> h=-1:1/18:1;
>> [H,T]=meshgrid(h,t);
>> Z=sign(H).*sqrt(1-R.^2);
>> surface(X,Y,Z),axis equal,view(3)
```



Siehe auch

`surf`, `text`, `shading`

Befehl

`light`

Anwendung

Eine oder mehrere Lichtquellen im Plot anbringen

Beschreibung

`light('position',[x y z])` plaziert an der Stelle (x,y,z) eine Lichtquelle, die den Lichtstrahl in den Achsenursprung (0,0,0) richtet.

`light('color',[rot grün blau])` definiert die Farbe des Lichtes. Die Skalare rot, grün und blau können Werte zwischen 0 und 1 annehmen.

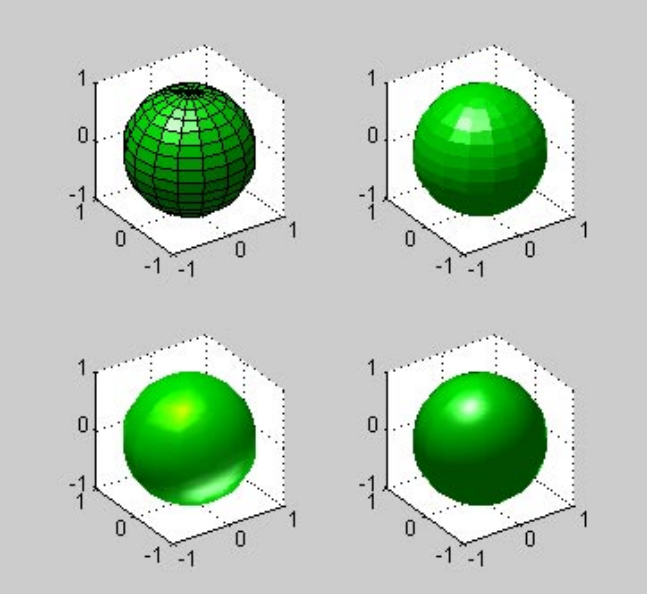
Mit `light('style','definition')` wird die Art der Lichtquelle bestimmt. Bei `infinite` (default-Einstellung) handelt es sich Quelle, die im Unendlichen liegt (parallele Strahlen), und bei `local` um eine Punktquelle.

Mit dem Befehl `lighting` wird die Beleuchtung von Objekten unterschiedlich berechnet. `lighting flat` berechnet für jede Teilfläche der Oberfläche die entsprechende Farbabstufung. `lighting gouraud` und `lighting phong` interpolieren die Farbabstufung über die ganze Fläche.

`lighting phong` liefert jedoch die besseren Resultate. `lighting none` schaltet die Beleuchtung aus.

Beispiel

```
>> subplot(2,2,1),sphere(16)
>> axis equal,colormap([0 1 0]),view(-35,30)
>> light('position',[-1 0 3])
>> subplot(2,2,2),sphere(16)
>> axis equal,view(-35,30), shading interp
>> light('position',[-1 0 3])
>> lighting flat
>> subplot(2,2,3),sphere(16)
>> axis equal,view(-35,30), shading interp
>> light('position',[-1 0 3],'color','y')
>> light('position',[1 -2 -4],'style','local')
>> lighting gouraud
>> subplot(2,2,4),sphere(16)
>> axis equal,view(-35,30), shading interp
>> light('position',[-1 0 3],'style','local')
>> lighting phong
```



Siehe auch

`lighting`, `material`, `surf`

4 Programmieren

Dieses Kapitel beinhaltet unter anderem Anweisungen, mit denen der Kontrollfluss in einem M-file gesteuert werden kann. Die Ausführung von Befehlen kann damit z.B. von logischen Bedingungen abhängig gemacht werden. Eine logische Bedingung, die wahr ist, hat in Matlab den Wert 1, eine nicht wahre Bedingung den Wert 0.

Folgende vier Auswahl- und Wiederholungsanweisungen stehen in MATLAB zur Verfügung:

1. `if`, zusammen mit `else` und `elseif`, führt eine Gruppe von Befehlen nur dann aus, wenn eine vorgegebene logische Bedingung erfüllt ist.
2. `for` durchläuft eine Schlaufe mit Befehlen für eine feste Anzahl von Wiederholungen.
3. `while` repetiert solange eine von einer logischen Bedingung abhängige Schlaufe, bis die logische Bedingung nicht mehr erfüllt ist.
4. `switch`, zusammen mit `case` und `otherwise`, führt abhängig von einer Entscheidungsvariablen unterschiedliche Gruppen von Befehlen aus.

Daneben werden die Möglichkeiten besprochen, Strings, die MATLAB-Befehle enthalten, auszuführen.

Ein ausführliches Unterkapitel ist dem Programmieren von Funktionen gewidmet.

Zum Schluss wird das Definieren von globalen Variablen behandelt.

4.1 Bedingte Befehlsabfolge

Befehle	Kurzbeschreibung
`if`	Logische Bedingung
`for`	Schlaufe mit vorgegebener Anzahl Durchläufe
`while`	Schlaufe mit logischer Abbruchbedingung
`switch`	Bedingte Verzweigung

Befehl

`if`

Anwendung

Abhängig von einer logischen Bedingung eine Gruppe von Befehlen ausführen

Beschreibung

Die allgemeine Form einer if-Anweisung lautet:

```
if expression1
   statements
elseif expression2
   statements
else
   statements
end
```

Falls der zu `if` gehörende Ausdruck, der eine logische Bedingung beschreibt, wahr (>0) ist, werden die darauf folgenden Befehle abgearbeitet. Dasselbe gilt für `elseif`. Ist keiner der Ausdrücke von `if` oder `elseif` wahr, werden die auf `else` folgenden Befehle ausgeführt. Es können mehrere `elseif` innerhalb der if-Anweisung vorkommen. `elseif` und `else` müssen aber nicht zwingend vorkommen. Wenn kein Ausdruck wahr ist, und kein `else` vorkommt, fährt MATLAB mit dem auf `end` folgenden Befehl fort. Für die logische Bedingung können die logischen Operatoren ==, <, >, <=, >= oder ~= verwendet werden.

M-File:Bspif.m

```
if x>0 & x<1
      disp('x ist grösser 0 aber kleiner 1.')
      xneu=x+1
      x=xneu;
  elseif x>=1 & x<=10
      disp('x liegt zwischen 1 und 10.')
      xneu=x-10
      x=xneu;
  elseif x>10
      disp('x ist grösser als 10.')
  elseif x==0
      disp('x ist 0.')
  else
      disp('x ist negativ.')
end
```

Beispiel

```
>> x=0.5;
>> Bspif
x ist grösser 0 aber kleiner 1.
xneu =
      1.5000
>> Bspif
x liegt zwischen 1 und 10.
xneu =
      -8.5000
>> Bspif
x ist negativ.
```

Siehe auch

`relop`

Befehl

`for`

Anwendung

Eine Schlaufe mit einer festen Anzahl Durchläufe

Beschreibung

Die allgemeine Form einer for-Schlaufe lautet:

```
for Index=Start:Inkrement:Ende
  Anweisungen
end
```

Index bezeichnet die Variable in der for-Schlaufe. Wird sie gestartet, so wird dem Index der Startwert zugewiesen. Bei jedem Durchlauf werden die Anweisungen in der for-Schlaufe ausgeführt, und der Index erhöht sich jeweils um den Wert des Inkrements bis der Endwert erreicht wird. Das Inkrement kann auch negativ sein. Das vordefinierte Inkrement hat den Wert 1.

Mit dem Befehl `break` kann aus einer Schlaufe vorzeitig ausgestiegen werden.

M-File:Bspfor1.m

```
for t=1:-0.2:0.2
  y(round(10*t/2))=exp(1/t);
end
```

M-File:Bspfor2.m

```
for i=1:n, for j=1:m, A(i,j)=i+j; end, end
```

Beispiel

```
>> Bspfor1;y
y =
      148.4132  12.1825  5.2945  3.4903  2.7183
>> n=1;m=3;
>> Bspfor2;A
A =
      2     3     4
      3     4     5
```

Siehe auch

`break`

Befehl

`while`

Anwendung

Abhängig von einer logischen Bedingung eine Schlaufe durchlaufen

Beschreibung

Die allgemeine Form einer while-Schlaufe lautet:

```
while Ausdruck
  Anweisungen
end
```

Solange die logische Bedingung wahr (Ausdruck > 0) ist, wird die Schlaufe durchlaufen, d.h. die Befehle werden ausgeführt. Für die logische Bedingung können die logischen Operatoren ==, <, >, <=, >= oder ~= verwendet werden.

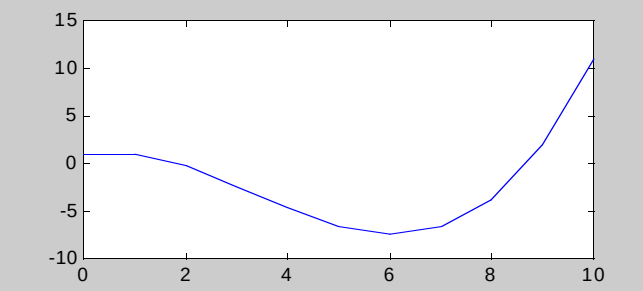
Mit dem Befehl `break` kann vorzeitig aus einer Schlaufe ausgestiegen werden.

M-File:Bspwhile.m

```
while n<=10
      y(n)=1+n-n^2+0.1*n^3;
      n=n+1;
end
```

Beispiel

```
>> n=0;Bspwhile;
>> x=0:10;plot(x,y)
```



Siehe auch

`break`

Befehl

`switch`

Anwendung

Unterschiedliche Teile eines M-Files ausführen, abhängig vom Wert, den ein festgelegter Ausdruck annimmt.

Beschreibung

Die allgemeine Form einer switch-Bedingung lautet:

```
switch Ausdruck
  case Wert1
    Anweisungen
  case {Wert2,Wert3,...}
    Anweisungen
  .
  .
  otherwise
    Anweisungen
end
```

`switch` vergleicht den Wert von "Ausdruck" mit den Werten in den verschiedenen `case`. Das erste `case`, von dem einer der Werte mit dem Wert von "Ausdruck" übereinstimmt, wird ausgeführt. Ein `case` kann mehrere Werte haben. Es wird maximal ein `case` ausgeführt. Stimmt keiner der Werte mit dem Wert von "Ausdruck" überein, so werden die unter `otherwise` aufgelisteten Befehle abgearbeitet, falls `otherwise` existiert.

Der Ausdruck bei `switch` kann ein Skalar oder ein String sein.

M-File:Bspswitch1.m

```
switch x
  case 1
    disp('x hat den Betrag 1')
  case {2,3,4}
    disp('x hat den Betrag 2,3 oder 4')
  otherwise
    disp('x hat weder den Betrag 1,2,3 noch 4')
end
```

M-File:Bspswitch2.m

```
Antwort=input('Möchten Sie einen Test machen?...
ja/nein  ','s');
disp(' ')
switch Antwort;
  case 'ja'
    disp('Wie vorbildlich!')
  case 'nein'
    disp('Wie schade!')
  otherwise
    disp('Ein "jein" kenne ich nicht!')
end
```

Beispiel

```
>> x=1;Bspswitch1
x hat den Betrag 1
>> x=-2;Bspswitch1
x hat weder den Betrag 1,2,3 noch 4
>> Bspswitch2
Möchten Sie einen Test machen? ja/nein  nein

Wie schade!
```

Siehe auch

`case`, `otherwise`

4.2 MATLAB Funktionen

Befehle	Kurzbeschreibung
`function`	Neue MATLAB Funktion definieren
`nargin`	Anzahl Eingabearg. einer MATLAB-Funktion
`nargout`	Anzahl Ausgabearg. einer MATLAB-Funktion
`global`	Globale Variable definieren

Befehl

`function`

Anwendung

Eine neue MATLAB-Funktion definieren

Beschreibung

Eine Funktion in MATLAB ist ein spezielles M-File, das durch den Befehl `function` in der ersten Zeile gekennzeichnet ist. Im Gegensatz zu Skript M-Files müssen bei Funktionen Variablen explizit übergeben und zurückgegeben werden. Skript M-Files arbeiten mit den Variablen aus dem MATLAB-Workspace. Funktionen dagegen haben ihren lokalen Workspace. D.h. in einer Funktion verwendete Variablen sind im MATLAB-Workspace nicht definiert und umgekehrt.

Eine Funktion besteht aus der Kopfzeile, dem help-Text und den Befehlszeilen.

Kopfzeile:

```
function y = fname(x)
```

Die Kopfzeile weist ein M-File als MATLAB-Funktion aus und legt die Syntax der Funktion fest. Die hier verwendeten Variablennamen müssen nicht identisch sein mit den später beim Funktionsaufruf verwendeten Variablennamen.

Die Kopfzeile muss mit dem Befehl `function` beginnen. Nach einem Leerzeichen werden die Ausgabeargumente definiert. Auf das Gleichheitszeichen folgt der Name der MATLAB-Funktion. Sie trägt bis auf die File-Erweiterung ".m" (die weggelassen wird) denselben Namen wie das M-File, in dem sie abgespeichert wird. Direkt nach dem Namen werden in einem Klammerpaar die Eingabeargumente definiert.

help-Text:

```
%    Es ist von Vorteil, eine Funktion mit
%    einem ausführlichen Kommentar zu ver-
%    sehen. Damit kann leichter nachvollzogen
%    werden, welchen Zweck die Funktion hat.
```

Auf die Kopfzeile folgt direkt der help-Text der Funktion, der durch %-Zeichen am Zeilenanfang als Kommentar gekennzeichnet ist. Wird in MATLAB `help fname` eingegeben, so wird dieser help-Text wiedergegeben. Die erste Zeile des help-Texts sollte den Funktionsnamen enthalten und mit einigen charakteristischen Begriffen den Zweck der Funktion umreissen. Der Befehl `lookfor` referenziert diese erste Zeile bei der Stichwortsuche in MATLAB.

Es erscheinen nur diejenigen Kommentare als help-Text, die zwischen der Kopfzeile und der ersten Befehlszeile liegen.

Befehlszeilen:

```
An dieser Stelle kann nun die eigentliche Funk-
tion eingegeben werden.
```

Die Funktion kann weiter Unterfunktionen, Schlaufen, Kalkulationen, Wertzuweisungen, Kommentare und Leerzeilen beinhalten oder andere Funktionen aufrufen.

Kommentar:

```
% beliebiger Text
```

Mit dem Befehl `return` kann eine Funktion vorzeitig verlassen werden, z.B. wenn eine vorgegebene Abbruchbedingung in der Funktion eingetreten ist.

Das Funktion-M-File "fname.m" sieht nun wie folgt aus:

```
function y = fname(x)
%FNAME H1 Zeile
%    help Zeilen
eigentliche Funktion % Kommentare
```

M-File:botta.m

```
function [V,AO,AM] = botta(r,s1,s2)
%BOTTA Schief abgeschnittenen Zylinder berechnen
%  Die Funktion botta(r,s1,s2) berechnet das Vol-
%  umen V, die Oberfläche AO und die Mantel-
%  fläche AM eines schief abgeschnittenen Zylin-
%  ders.
%  r ist der Radius, s1 die längste Mantellinie
%  und s2 die kürzeste.
V=pi*r^2*(s1+s2)/2;
AO=pi*r*(s1+s2+r+sqrt(r^2+(s1+s2)^2/4));
AM=pi*r*(s1+s2);
```

Beispiel

```
>> help botta
 BOTTA Schief abgeschnittenen Zylinder berechnen
   Die Funktion botta(r,s1,s2) berechnet das Vol-
   umen V, die Oberfläche AO und die Mantel-
   fläche AM eines schief abgeschnittenen Zylin-
   ders.
   r ist der Radius, s1 die längste Mantellinie
   und s2 die kürzeste.
>> [Volumen,Oberfläche,Mantelfläche]=botta(4,5,2)
Volumen =
      175.9292
Oberfläche =
      205.0213
Mantelfläche =
      87.9646
```

Siehe auch

`script`, `return`, `inputname`

Befehl

`nargin`

Anwendung

Die Anzahl der Eingabeargumente einer Funktion bestimmen

Beschreibung

In einer Funktion eruiert der Befehl `nargin`, wieviele Argumente die Funktion erhalten hat. Je nach Anzahl der Argumente können dann z.B. unter Verwendung von if-Anweisungen unterschiedliche Aufgaben bzw. Berechnungen durchführt werden.

`nargin('fname')` gibt die Anzahl der definierten Eingabeargumente der Funktion “fname”.

M-File:Bspnargin.m

```
function y = Bspnargin(a,b)
if nargin<1
  error('Keine Eingänge')
elseif nargin==1
  y=a^2-4;
elseif nargin==2
  y=a^2+4*b^2-4;
end
```

Beispiel

```
>> Bspnargin
??? Error using ==> bspnargin
Keine Eingänge
>> Bspnargin(3)
ans =
      5
>> Bspnargin(3,2)
ans =
      21
```

Siehe auch

`varargin`, `varargout`

Befehl

`nargout`

Anwendung

Die Anzahl Ausgabeargumente einer Funktion bestimmen

Beschreibung

In einer Funktion eruiert der Befehl `nargout`, wieviele Werte die Funktion auszugeben hat. Je nach Anzahl der verlangten Ausgabeargumente können dann z.B. mittels einer if-Anweisung unterschiedliche Befehle zur Ausführung gelangen.

`nargout('fname')` gibt die Anzahl der definierten Ausgabeargumente der Funktion “fname” an.

M-File:Bspnargout.m

```
function [y,z] = Bspnargout(a,b)
if nargin<1
  error('Keine Eingänge')
elseif nargin==1
  y=a^2-4;
 elseif nargin==2
  y=a^2+4* b^2-4;
  if nargout==2
    z=a^2+b-16;
  end
end
```

Beispiel

```
>> [y,z]=Bspnargout(3,2)
y =
      21
z =
      -5
>> Bspnargout(3,2)
ans =
      21
>> [y,z]=Bspnargout(2)
Warning: One or more output arguments not
assigned during call to 'bspnargout'.
y =
      0
```


Befehl

`global`

Anwendung

Eine globale Variable definieren

Beschreibung

Grundsätzlich sind die in einer Funktion verwendeten Variablen nur lokal definiert. Aus einer anderen Funktion oder aus dem Workspace kann nicht darauf zugegriffen werden. Wie auch die Funktion selber nicht auf Variablen des workspace oder anderer Funktionen zugreifen kann.

Wenn aber in einer Funktion eine Variable als globale Variable definiert wird, ist sie für alle anderen Funktionen und für den workspace zugänglich, sofern sie dort ebenfalls für `global` erklärt wurde.

M-File:Bspglobal.m

```
function y = Bspglobal(a)
global b % globale Variable b
if nargin<1
  error('Keine Eingänge')
elseif nargin==1
  y=b^2+a^2-4;
  b=b+1;
end
```

Beispiel

```
>> global b % globale Variable b
>> b
b =
      []
>> b=3;
>> Bspglobal(3)
ans =
      14
>> b
>> b =
      4
```

Siehe auch

`isglobal`, `clear`, `who`

4.3 Befehle auswerten und ausführen

Befehle	Kurzbeschreibung
`eval`	MATLAB-Befehl ausführen
`feval`	MATLAB Funktion ausführen
`run`	Ein M-File mit Befehlen starten

Befehl

`eval`

Anwendung

Einen in einem String enthaltenen MATLAB-Befehl ausführen

Beschreibung

`eval('Befehl')` interpretiert den String als MATLAB-Befehl und behandelt ihn dementsprechend.

Mit `eval('Befehl1','Befehl2')` besteht die Möglichkeit, Fehlermeldungen zu unterdrücken, indem in zwei Strings für eine Berechnung zwei unterschiedliche Befehle eingegeben werden. Falls einer der beiden Befehle einen Fehler erzeugt, wird ohne Fehlermeldung der andere ausgeführt. Im ersten String kann z.B. eine neue Berechnung ausprobiert werden, und im zweiten der alte Befehl eingegeben werden. Damit ist eine Ausgabe garantiert.

Beispiel

```
>> A=[1 2 3;4 5 6];
>> eval('A*A')
??? Error using ==> *
Inner matrix dimensions must agree.
>> Atransp=A'
>> eval('A*A','A*Atransp')
ans =
      14    32
      32    77
```

Siehe auch

`assignin`

Befehl

`feval`

Anwendung

Eine in einem String enthaltene MATLAB-Funktion ausführen

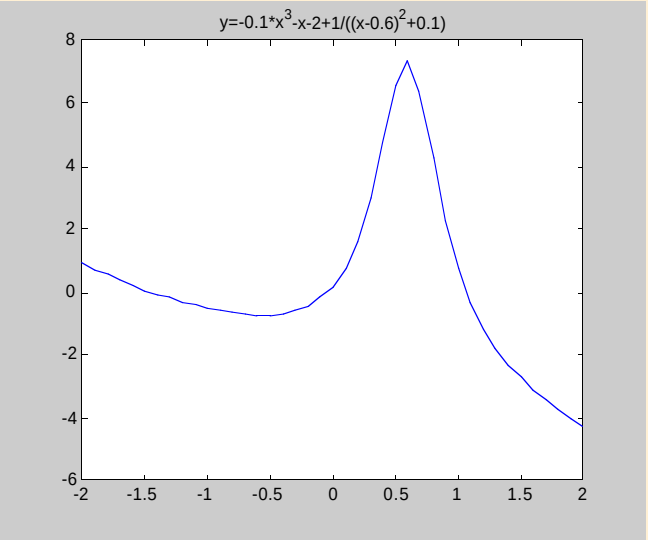
Beschreibung

`feval('Funktion,x1,...,xn')` interpretiert den String als MATLAB-Funktion und behandelt ihn dementsprechend. `feval` berechnet die genannte Funktion, die gewöhnlich in einem separaten M-File definiert wird, an den Stellen x1 bis xn.

Zum Beispiel berechnet `feval('huch',-2)` die Funktion “huch” (siehe `fmin`) an der Stelle x = –2. Dies entspricht dem Befehl `huch(-2)`.

Beispiel

```
>> t=-2:0.1:2;
>> plot(t,feval('huch',t))
```



Siehe auch

`builtin`

Befehl

`run`

Anwendung

Ein M-File ausführen, das nicht in einem Directory des aktuellen Pfads gespeichert ist.

Beschreibung

Normalerweise wird der Name eines M-Files im Command Window eingetippt, um es auszuführen. Dies funktioniert jedoch nur, wenn das das M-File enthaltende Directory im aktuellen Pfad vermerkt ist.

Mit dem Befehl `run` kann nun ein M-File ausgeführt werden, das sich nicht im aktuellen Pfad befindet.

Mit `run Dateinamen` (Dateinamen schliesst den Pfadnamen mit ein) wird das entsprechende M-File gestartet. Wird im Dateinamen der komplette Pfadname angegeben, so wechselt `run` vom momentan aktiven Directory zu dem Directory, in dem sich das betreffende M-File befindet, führt es aus, und wechselt wieder ins ursprünglich aktive Directory zurück.

Siehe auch

`cd`,`addpath`

5 Differentialrechnung, Integralrechnung und Differentialgleichungen

Dieses Kapitel behandelt das Differential- und Integralrechnen. Von einer Funktion n-ter Ordnung werden die Minima, die Nullstellen und das bestimmte Integral berechnet. Die Funktionen n-ter Ordnung werden zu diesem Zweck in MATLAB Funktionen definiert. Im Kapitel “Programmieren” wird unter function näher auf die Programmierung von MATLAB Funktionen eingegangen.

Im Unterkapitel “Differentialgleichungen” wird das Lösen von Differentialgleichungen im MATLAB besprochen.

5.1 Differential- und Integralrechnung

5.1.1 Minima und Nullstellen

Befehle	Kurzbeschreibung
`fmin`	Minimum einer Funktion mit einer Variable
`fmins`	Minimum einer Funktion mit zwei Variablen
`fzero`	Nullstelle einer Funktion mit einer Variable

Befehl

`fmin`

Anwendung

Lokale Minima einer Funktion mit einer Variable

Beschreibung

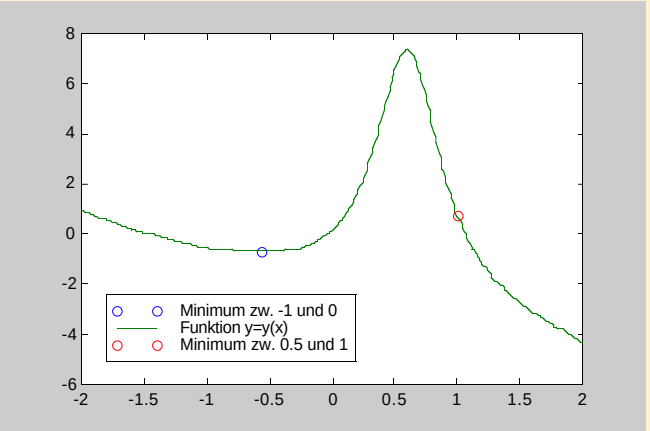
`x=fmin('f',x1,x2)` sucht ein lokales Minimum der Funktion f(x) im Intervall $x_1<x<x_2$. `'f'` ist ein String und steht für den Namen der in einem M-File programmierten MATLAB Funktion. Die einfachste Art, eine neue MATLAB Funktion zu programmieren, wird kurz anhand eines Beispiels erläutert:

```
function y = huch(x)
%HUCH berechnet eine Funktion 3. Ordnung.
% Die Funktion 3. Ordnung lautet:
% y=-0.1*x^3-x-2+1/((x-0.6)^2+0.1)
% Als Eingang hat huch den Vektor (Skalar) x
% x beschreibt das Intervall, für welches die
% Funktion y ermittelt wird. y ist der Ausgang
% der “function” huch.m.
y=-0.1.*x.^3+1./((x-0.6).^2+0.1)-x-2;
```

Mit `x=fmin('f',x1,x2,options)` können zusätzliche Einstellungen vorgenommen werden, wie z.B. die Zwischenschritte der Minimierung editieren, die Toleranz von x verändern, usw. (siehe -> `help fmin`).

Beispiel

```
>> x=-2:0.01:2;y=huch(x);
>> % lokales Minimum zwischen -1 und 0 und
>> % der option(1)=1
>> xmin1=fmin('huch',-1,0,1)
 Func evals     x          f(x)         Procedure
    1      -0.618034    -0.726889        initial
    2      -0.381966    -0.672839        golden
    3      -0.763932    -0.681362        golden
    4      -0.580826    -0.730392        parabolic
    5      -0.559878    -0.730682        parabolic
    6      -0.565337    -0.730728        parabolic
    7      -0.565544    -0.730728        parabolic
    8      -0.565508    -0.730728        parabolic
    9      -0.565474    -0.730728        parabolic
xmin1 =
      -0.5655
>> ymin1=huch(xmin1);
>> % lokales Minimum zwischen 0.5 und 1
>> xmin2=fmin('huch',0.5,1) xmin2 =
      0.9999
>> ymin2=huch(xmin2);
>> plot(xmin1,ymin1,'o',x,y,xmin2,ymin2,'ro')
>> legend('Minimum zw. -1 und 0',...
   'Funktion y=y(x)','Minimum zw. 0.5 und 1')
```



Befehl

`fmins`

Anwendung

Lokale Minima einer Funktion mit zwei oder mehr Variablen

Beschreibung

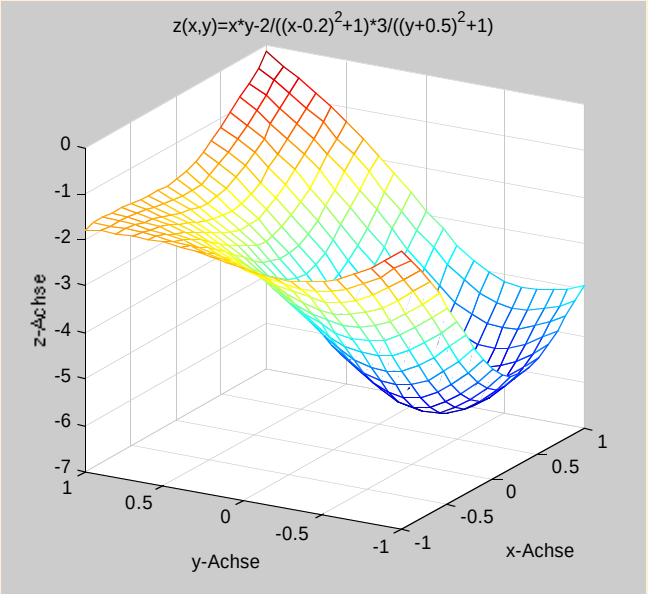
`X=fmins('F',X0)` sucht ein lokales Minimum der Funktion F(x) nahe dem Startvektor X_0. `'F'` ist ein String und steht für den Namen der in einem M-File programmierten MATLAB Funktion. Für das Beispiel wird folgende neue MATLAB Funktion programmiert:

```
function z = blabla(r);
%BLABLA berechnet die Funktion z.
% Die Funktion z lautet:
% z(x,y)=x*y-2/((x-0.2)^2+1)*3/((y+0.5)^2+1).
% Als Eingang hat blabla die Matrix (Vektor) r.
% Sie hat zwei Kolonnen für die x- und y-Koordi-
% naten.
% z ist der Ausgang der “function” blabla.m.
x=r(1);y=r(2);
z=x.*y-2./((x-0.2).^2+1).*3./((y+0.5).^2+1);
```

Bei `X=fmin('F',X0,options)` können wiederum mit dem Vektor “options” zusätzliche Einstellungen vorgenommen werden. Sie entsprechen denjenigen von `fmin`.

Beispiel

```
>> r=[-1 0]; % Startwert
>> zmin=fmins('blabla',r) % Minimum von z(x,y)
zmin =
      0.2435   -0.5204
>> t=-1:0.1:1;
>> [x,y]=meshgrid(t,t);
>> z=x.*y-2./((x-0.2).^2+1).*3./((y+0.5).^2+1);
>> mesh(x,y,z)
>> title(['z(x,y)=x*y-2/((x-0.2)^2+1)*3/',...
   '((y+0.5)^2+1)'])
>> xlabel('x-Achse')
>> ylabel('y-Achse')
>> zlabel('z-Achse')
>> view(-60,20)
```



Siehe auch

`foptions`

Befehl

`fzero`

Anwendung

Nullstellen einer Funktion mit einer Variable

Beschreibung

`fzero('f',x)` ermittelt die Nullstellen der Funktion f(x). `'f'` ist ein String und steht für den Namen der in einem M-File programmierten MATLAB Funktion. Die Variable x ist entweder ein Skalar, oder ein Vektor der Länge 2. Den Skalar x betrachtet `fzero` als Startwert. Von diesem Punkt aus sucht MATLAB die Funktion f(x) nach einer Nullstelle ab. Der Vektor x =[x_1,x_2] definiert ein Intervall auf der x-Achse, in welchem die Nullstellen der Funktion f(x) eruiert werden. x_1 ist der Start- und x_2 der Endwert.

Für das Beispiel wird die weiter oben definierte MATLAB Funktion “huch” verwendet.

`fzero('f',x,tol)` setzt für die Konvergenz die relative Toleranz “tol”.

`fzero('f',x,tol,trace)` informiert über jede Iteration, falls `trace` nicht Null ist.

Beispiel

```
>> xnull1=fzero('huch',[-2 -1],[],1)
xnull1 =
      -1.4590
>> xnull2=fzero('huch',-0.5)
xnull2 =
      -0.0406
```

Siehe auch

`roots`

5.1.2 Numerische Integration

Befehle	Kurzbeschreibung
quad	Integration mit der Simpson Regel
quad8	Integration mit der Newton Regel

Befehl

`quad`

Anwendung

Numerische Integration, Methode niedriger Ordnung

Beschreibung

`q=quad('f',a,b)` berechnet das bestimmte Integral der Funktion f(x). f ist der Integrand, x die Integrationsvariable, a die untere und b die obere Integrationsgrenze, wobei a<b. `quad` verwendet für die numerische Integration der Funktion f(x) die adaptive, rekursive Simpson Regel. Der relative Fehler liegt bei `1e-3`. q entspricht der Fläche unterhalb der Kurve zwischen a und b. `'f'` ist ein String und steht für den Namen der in einem M-File programmierten MATLAB Funktion.

Für das Beispiel wurde folgende neue MATLAB Funktion programmiert:

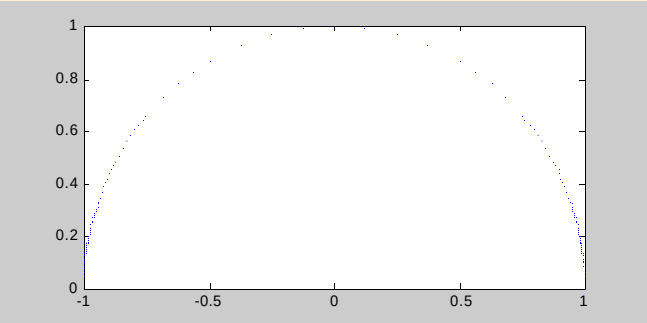
```
function f = kreis(t)
%KREIS berechnet einen Halbkreis
%  Die “function” kreis(t) berechnet die Funktion
%  f(t)=sqrt(1-t.^2).
%  Im Intervall -1≤t≤1 werden für den Halbkreis
%  mit dem Radius 1 die Werte für die Ordinate
%  ermittelt.
%  Als Eingang erhält kreis(x) die x-Koordinaten.
%  Als Ausgang liefert f(t) die y-Koordinaten des
%  Kreises.
f=sqrt(1-t.^2);
```

Mit `q=quad('f',a,b,tol)` integriert `quad` mit einem relativen Fehler der Grösse tol. `tol=[rel_tol abs_tol]` definiert den absoluten Fehler.

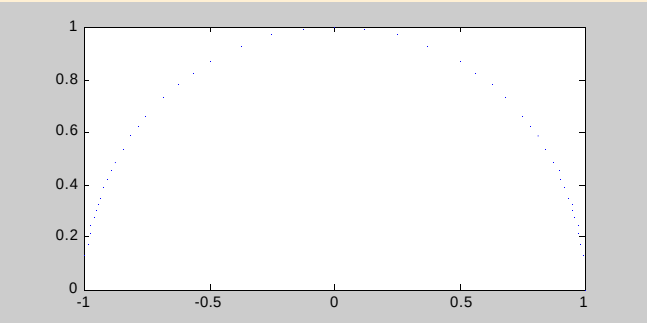
`q=quad('f',a,b,tol,trace)` editiert jeden Iterationsschritt, falls `trace` nicht Null ist.

Beispiel

```
>> LängeHalbkreis=pi/2
LängeHalbkreis =
>> 1.5708
>> Fläche=quad('kreis',-1,1,[],1)
Warning: Recursion level limit reached 16 times.
Fläche =
      1.5708
```




```
>> Fläche=quad('kreis',-1,1,1e-1)
Fläche =
      1.5418
>> Fläche=quad('kreis',-1,1,[1e-4 1e-2],1)
Fläche =
      1.5706
```



Siehe auch

`dblquad`

Befehl

`quad8`

Anwendung

Numerische Integration, Methode hoher Ordnung

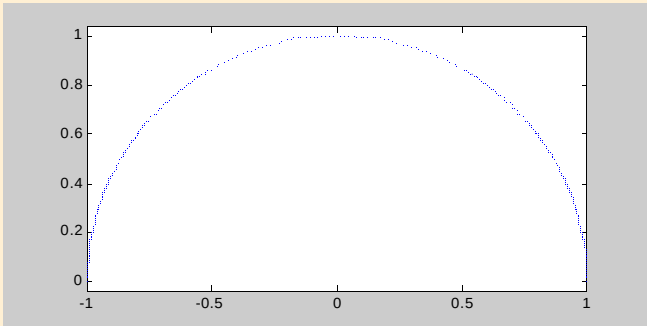
Beschreibung

`q=quad8('f',a,b)` berechnet das bestimmte Integral der Funktion f(x). `quad` und `quad8` unterscheiden sich nur in der Integrationsmethode. `quad8` verwendet für die numerische Integration der Funktion f(x) eine adaptive, rekursive Newton Regel.

Für das Beispiel wird die MATLAB Funktion “kreis” aus `quad` verwendet.

Beispiel

```
>> format long
>> Halbkreisfläche=pi/2
Halbkreisfläche =
      1.57079632679490
>> Fläche=quad8('kreis',-1,1)
Fläche =
      1.57079610082586
>> Fläche=quad8('kreis',-1,1,1e-10,1)
Fläche =
      1.57079610082794
```



5.1.3 Polynome plotten

Befehle	Kurzbeschreibung
`ezplot`	Funktion plotten

Befehl

`ezplot`

Anwendung

Eine Funktion graphisch darstellen

Beschreibung

`ezplot('f')` zeichnet in einem Graphik-Fenster die Funktion f(x). `'f'` ist ein String und steht für den Namen der in einem M-File programmierten MATLAB Funktion.

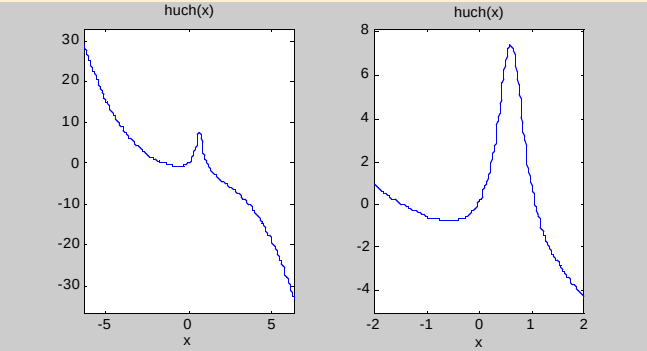
Als Eingabeargument benötigt `ezplot` den Namen der MATLAB Funktion und in Klammern ihre Variable. Das default-Intervall für die Abszisse ist `[-2*pi, 2*pi]`.

Für die MATLAB Funktion “kreis.m” (siehe -> `quad`) zeichnet `ezplot('kreis(t)')` einen Halbkreis im Intervall [–1 1].

`ezplot('f',xmin,xmax)` plottet im Intervall [x_{min} x_{max}].

Beispiel

```
>> subplot(1,2,1),ezplot('huch(x)')
>> subplot(1,2,2),ezplot('huch(x)',-2,2)
```



5.2 Differentialgleichungen

Befehle	Kurzbeschreibung
`ode45`	gewöhnliche Differentialgleichungen lösen
`ode23`	gewöhnliche Differentialgleichungen lösen

Befehl

`ode45`

Anwendung

Lösen gewöhnlicher Differentialgleichungen, Methode mittlerer Ordnung

Beschreibung

`ode` steht für “ordinary differential equation”. Für eine gewöhnliche Differentialgleichung gilt:

$$y^{(n)} = f(t, y, y', ..., y^{(n-1)}) \quad (14)$$

Bei einer Anfangswert-Aufgabe sind folgende Anfangsbedingungen zu erfüllen:

$$y(t_0) = a_1, y'(t_0) = a_2, y''(t_0) = a_3, ..., y^{(n-1)}(t_0) = a_n \quad (15)$$

Die gewöhnliche Differentialgleichung hat genau eine Lösung y=g(x) mit

$$g(t_0) = a_1, g'(t_0) = a_2, g''(t_0) = a_3, ..., g^{(n-1)}(t_0) = a_n\ , \quad (16)$$

falls die gewöhnliche DGL in der Umgebung des Anfangswertes stetig ist und dort stetige partielle Ableitungen 1. Ordnung nach y, y', y'', ..., $y^{(n-1)}$ besitzt.

Im MATLAB wird die gewöhnliche Differentialgleichung n-ter Ordnung mittels Substitution in ein System von n linearen Differentialgleichungen 1. Ordnung umgewandelt:

$$y_1 = y, y_2 = y', y_3 = y'', ..., y_n = y^{(n-1)} \quad (17)$$

Daraus folgt:

$y'_1 = y_2, y'_2 = y_3, ..., y'_n = f(t, y_1, y_2, y_3, ..., y_n)$ (18)

Z.B. lautet die gewöhnliche DGL für das gedämpfte Feder-Masse-System: y"+(k/m)y'+(c/m)y = 0 mit y(0) = 0 und y'(0) = 0.

Mit der Substitution $y_1 = y$ und $y_2 = y'$ hat die DGL des gedämpften Feder-Masse-Systems die Form y'=f(t,y), $y(t_0)=y_0$:

$y'_1 = y_2$ und $y'_2 = -(k/m)y_2-(c/m)y_1$ (19)

Für dieses Beispiel wurde folgende neue MATLAB Funktion programmiert:

```
function dy = FederMasse(t,y)
%FEDERMASSE löst die DGL des Feder-Masse-Systems
% FederMasse(t,y) beschreibt das gedämpfte Feder-
% Masse-System y''+δ*y'+ω^2*y=0. δ=k/2m ent-
% spricht der Dämpfung und hat den Betrag 0.5.
% ω^2=c/m steht für die Kreisfrequenz mit dem
% Betrag 1. FederMasse(t,y) hat zwar zwei Ein-
% gänge t und y, muss sie aber nicht verwenden.
% Der Ausgang ist der Kolonnenvektor dy.
dy=[y(2);-0.5*y(2)-y(1)];
```

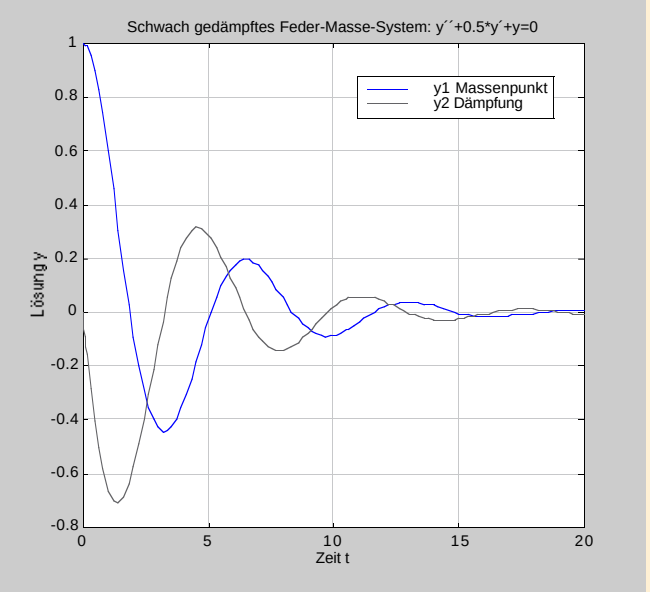
`[T,Y]=ode45('f',tspan,y0)` integriert das System von Differentialgleichungen y'=f(t,y) im Zeitintervall `tspan` mit den Anfangswerten y_0. `'f'` ist ein String und steht für den Namen der in einem M-File programmierten MATLAB Funktion. `tspan` definiert das Zeitintervall [t_0 t_{ende}] und y_0 die Anfangswerte zum Zeitpunkt t_0.

Jeder Wert aus dem Kolonnenvektor Y ist in Funktion der Zeit T, d.h. zu jedem Wert aus T gehört ein Wert aus Y.

Bei `[T,Y]=ode45('f',tspan,y0,options)` können unter `options` andere Integrationsparameter gesetzt werden (siehe -> `help ode45` oder `help odeset`).

Beispiel

```
>> [T,Y]=ode45('FederMasse',[0 20],[1 0]);
>> plot(T,Y(:,1),'-',T,Y(:,2),'--')
>> title(['Schwach gedämpftes Feder-Masse-',...
   'System: y´´+0.5*y´+y=0'])
>> xlabel('Zeit t'),ylabel('Lösung y')
>> legend('y1 Massenpunkt','y2 Dämpfung')
>> grid on
```



Siehe auch

`ode15s`, `ode23s`, `odeset`, `odeget`, `odeplot`, `odeprint`

Befehl

`ode23`

Anwendung

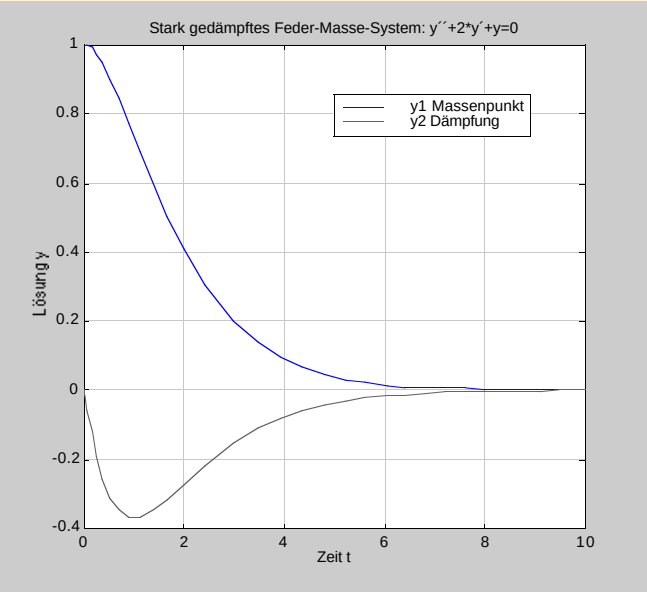
Lösen gewöhnlicher Differentialgleichungen, Methode niedriger Ordnung

Beschreibung

`ode23` löst wie `ode45` ein System von gewöhnlichen Differentialgleichungen. Die Integrationsmethode ist jedoch niedriger Ordnung.

Beispiel

```
>> [T,Y]=ode23('FederMasse',[0 10],[1 0]);
>> plot(T,Y(:,1),'-',T,Y(:,2),'--')
>> title(['Stark gedämpftes Feder-Masse-',...
   'System: y´´+2*y´+y=0'])
>> xlabel('Zeit t'),ylabel('Lösung y')
>> legend('y1 Massenpunkt','y2 Dämpfung')
```



Siehe auch

`ode15s`, `ode23s`

6 Datenanalyse und Statistik

Bei Messungen entstehen Daten. Um daraus jedoch verwendbare Informationen zu gewinnen, sind meist noch weitere Bearbeitungsschritte nötig. In diesem Kapitel sehen Sie, wie Sie Daten in MATLAB einlesen und mit statistischen Methoden analysieren können.

6.1 File Input/Output

6.1.1 Files öffnen und schliessen

Befehle	Kurzbeschreibung
`fopen`	File öffnen
`fclose`	File schliessen

Befehl

`fopen`

Anwendung

Ein ASCII- oder Binär-File öffnen

Beschreibung

Sollen Daten aus einem ASCII- oder Binär-File gelesen werden oder in ein solches File geschrieben werden, so muss es zuerst mit `fopen` geöffnet werden.

`fid=fopen('myfile','perm')` öffnet das File “myfile”. Mit dem String `'perm'` werden die Berechtigungen für den File-Zugriff festgelegt:

- `'r'` lesen
- `'w'` schreiben
- `'a'` am Ende hinzufügen (und File erzeugen, wenn nötig)
- `'r+'` lesen und schreiben

Dabei spielt es unter Unix und auf dem Macintosh keine Rolle, ob es sich um ein Text- oder ein Binär-File handelt. Auf dem PC hingegen muss der Buchstabe `'t'` hinzugefügt werden, wenn es sich um ein Text-File handelt, also z.B. `'rt'` zum Lesen.

Falls ein File erfolgreich geöffnet werden konnte, wird der Variablen fid eine positive Zahl zugewiesen. Sie entspricht einer File-Identifikationszahl, die bei weiteren File-Input/Output Befehlen wie z.B. `fread`, `fwrite` oder `sprintf` anstelle des Filenamens verwendet wird.

Konnte das File nicht geöffnet werden, so wird fid der Wert `-1` zugewiesen.

`[fid,message]=fopen('filename','perm')`

liefert die File-Identifikationszahl fid und — falls das Öffnen nicht erfolgreich war — die Fehlermeldung message.

Beispiel

```
>> fid=fopen('muster.dat','r')
fid =
      -1
>> [fid,message] = fopen('muster.dat','r')
fid =
      -1
message =
Cannot open file. Existence? Permissions? Memory?
>> fid = fopen('/dev/rst0','r+')
fid =
      3
```

Siehe auch

`fread`, `fwrite`, `fscanf`, `fprintf`, `sprintf`, `sscanf`

Befehl

`fclose`

Anwendung

Ein ASCII- oder Binär-File schliessen

Beschreibung

`st=fclose(fid)` schliesst das File mit der Identifikationszahl fid. Wenn das File erfolgreich geschlossen werden konnte, so wird der Statusvariablen st der Wert `0` zugewiesen, wenn nicht, dann erscheint eine Fehlermeldung.

`fclose('all')` schliesst alle offenen Dateien, ausser den speziellen Dateien mit fid = 0 (Standard-Input), fid = 1 (Standard-Output) und fid = 2 (Standard-Error).

Beispiel

```
>>st=fclose('all')
st =
     0
```

Siehe auch

`fread`, `fwrite`, `fscanf`, `fprintf`, `sprintf`, `sscanf`

6.1.2 Dateneingabe und -ausgabe im Command Window

Befehle	Kurzbeschreibung
`clc`	Die Anzeige im Command Window löschen
`disp`	Eine Matrix oder einen Text anzeigen
`input`	Eingabe verlangen
`pause`	Programm unterbrechen

Befehl

`clc`

Anwendung

Alles löschen, was im Command Window zu sehen ist

Beschreibung

`clc` löscht alle im Command Window sichtbaren Befehle und Resultate und setzt den Cursor in die linke obere Fensterecke.

Die Variablen werden bei `clc` nicht gelöscht.

Befehl

`disp`

Anwendung

Im Command Window eine Matrix oder einen Text anzeigen

Beschreibung

`disp(X)` zeigt die Werte der Matrix X, ohne jedoch ihren Namen anzuzeigen. Mit `disp('mytext')` wird der String "mytext" angezeigt.

Beispiel

```
>> A=[1 2;3 4]
A =
      1     2
      3     4
>> disp(A)
      1     2
      3     4
>> C='Dies ist ein String';
>> disp(C)
      Dies ist ein String
```

Siehe auch

`int2str`, `num2str`, `sprintf`, `rats`, `format`

Befehl

`input`

Anwendung

Wartet auf eine Eingabe

Beschreibung

`r=input('question')` stellt die Frage “question” und wartet auf eine Eingabe, die durch Drücken der Return-Taste abgeschlossen wird. Als Eingabe kann ein beliebiger MATLAB-Ausdruck verwendet werden (Matrix, Vektor, Skalar, Variablenname etc.), dessen Wert dann der Variablen r zugewiesen wird.

`r=input('question','s')` wartet auf die Eingabe eines Strings.

Beispiel

```
>> r=input('Wann wurde die ETH gegründet?')
Wann wurde die ETH gegründet?1854
r =
      1854
q='Möchten Sie die Übung fortsetzen?     y/n   ';
>> r=input(q,'s')
Möchten Sie die Übung fortsetzen?     y/n   y
r =
      y
```

Siehe auch

`keyboard`

Befehl

`pause`

Anwendung

Das laufende Programm unterbrechen und auf einen Tastendruck warten

Beschreibung

`pause(n)` wartet n Sekunden; danach wird das gerade ausgeführte Programm fortgesetzt.

`pause` wartet mit der Fortsetzung der Berechnung solange, bis eine Taste gedrückt wird.

`pause` wird oft bei Graphiken verwendet, damit der Benützer genügend Zeit zum Betrachten hat.

`pause off` deaktiviert alle folgenden `pause`- oder `pause(n)`-Befehle. Mit `pause on` werden alle folgenden `pause`- oder `pause(n)`-Befehle (wieder) aktiviert.

6.2 Statistik

6.2.1 Grundoperationen

Befehle	Kurzbeschreibung
`max`	Maximum
`mean`	Mittelwert
`std`	Standardabweichung
`sort`	Sortieren
`sum`	Summe
`prod`	Produkt

Befehl

`max`

Anwendung

Bestimmt das Maximum eines Vektors oder einer Matrix

Beschreibung

Bei einem Vektor X bestimmt `max(X)` das grösste Element.

Ist X eine Matrix, so liefert `max(X)` einen Zeilenvektor, der jeweils das grösste Element jeder einzelnen Kolonne von X enthält.

`[Y,I]=max(X)` weist der Variablen Y das Maximum von X zu. Der Zeilenvektor I zeigt, welches Element (bei einem Vektor X) bzw. welche Zeilen (bei einer Matrix X) die Maxima enthalten.

Bei `Z=max(X,Y)` werden bei den Matrizen X und Y elementweise die jeweiligen Maxima bestimmt und der Matrix `Z` zugewiesen, die somit dieselben Dimensionen wie X und Y besitzt.

Für den Befehl `min` gelten dieselben Regeln.

Beispiel

```
>> A=[5 1 8;6 9 2;7 2 5];
>> max(A)
ans =
      7     9     8
>> [Y,I]=max(A)
Y =
      7     9     8
I =
      3     2     1
>> B=[1 3 9;7 2 1;7 6 2];
>> max(A,B)
ans =
      5     3     9
      7     9     2
      7     6     5
>> [Y,I]=min(A)
Y =
      5     1     2
I =
      1     1     2
```

Siehe auch

`min`, `median`

Befehl

`mean`

Anwendung

Den Mittelwert aus einer Menge von Werten berechnen

Beschreibung

`mean(X)` berechnet den Mittelwert der Elemente des Vektors X. Handelt es sich bei X um eine Matrix, so steht in jedem Element des resultierenden Zeilenvektors der Mittelwert des entsprechenden Kolonnenvektors von X.

Für `mean(X,1)` werden die Mittelwerte kolonnenweise (also genau wie bei `mean(X)`) und für `mean(X,2)` zeilenweise berechnet.

Beispiel

```
>> A=[5 1 8;6 9 2;7 2 5]
A =
      5     1     8
      6     9     2
      7     2     5
>> mean(A)
ans =
      6     4     5
>> mean(A,1)
ans =
      6     4     5
ans =
      4.6667
      5.6667
      4.6667
```

Siehe auch

`median`, `cov`

Befehl

`std`

Anwendung

Die Standardabweichung berechnen

Beschreibung

Die Standardabweichung s eines Datenvektors X kann folgendermassen berechnet werden:

$$s = \sqrt{\frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right)} \qquad (20)$$

`s=std(X)` bestimmt die Standardabweichung der Elemente des Vektors X. Handelt es sich bei X um eine Matrix, so ist s ein Zeilenvektor, bei dem jedes Element für die Standardabweichung der entsprechenden Kolonne der Matrix X steht.

`std(X)` normiert mit n-1 Messwerten. Handelt es sich bei X um eine Zufallsstichprobe aus normalverteilten Daten, so entspricht `std(X).^2` der besten erwartungstreuen Schätzung für die Varianz.

`s=std(X,flag)` für flag = 0 ist dasselbe wie `std(X)`. Für flag = 1 liefert `std(X,1)` die Standardabweichung normiert mit n Messwerten (zentrales Moment zweiter Ordnung).

Für `std(X,flag,1)` werden die Standardabweichungen kolonnenweise und für `std(X,flag,2)` zeilenweise berechnet.

Beispiel

```
>> A=[5 1 8;6 9 2;7 2 5];
>> std(A)
ans =
      1.0000    4.3589    3.0000
>> std(A,0,1),std(A,1,1)
ans =
      1.0000    4.3589    3.0000
ans =
      0.8165    3.5590    2.4495
```

Siehe auch

`cov`, `corrcoef`

Befehl

`sort`

Anwendung

Elemente nach der Grösse ordnen

Beschreibung

`sort(X)` ordnet die Elemente des Vektors X nach der Grösse, so dass das kleinste Element am Anfang steht und das grösste am Schluss. Handelt es sich bei X um eine Matrix, so wird jede Kolonne für sich nach der Grösse sortiert.

Für `sort(X,1)` wird kolonnenweise sortiert (ist also dasselbe wie `sort(X)`) und für `sort(X,2)` zeilenweise.

`[Y,I]=sort(X)` ordnet X nach der Grösse und speichert das Resultat in Y ab. Die Index-Matrix I hat dieselben Dimensionen wie X und zeigt, an welchen Positionen von X die Elemente von Y vor dem Sortieren standen (siehe Beispiel).

Beispiel

```
>> A=[5 1 8;6 9 2;7 2 5];
>> [Y,I]=sort(A,1)
Y =
     5     1     2
     6     2     5
     7     9     8
I =
     1     1     2
     2     3     3
     3     2     1
>> [Y,I]=sort(A,2)
Y =
     1     5     8
     2     6     9
     2     5     7
I =
     2     1     3
     3     1     2
     2     3     1
```

Siehe auch

`sortrows`

Befehl

`sum`

Anwendung

Die Summe der einzelnen Elemente

Beschreibung

`sum(X)` berechnet die Summe der Elemente des Vektors X. Handelt es sich bei X um eine Matrix, so wird die Summe über jede einzelne Kolonne berechnet und daraus ein Zeilenvektor gebildet.

`sum(X,dim)` summiert entlang der Dimension dim, d.h. für `sum(X,1)` wird kolonnenweise summiert und für `sum(X,2)` zeilenweise.

Beispiel

```
>> A=[5 1 8;6 9 2;7 0 5]
A =
     5     1     8
     6     9     2
     7     0     5
>> sum(A,1)
ans =
     18    10    15
>> sum(A,2)
ans =
     14
     17
     12
```

Siehe auch

`cumsum`

Befehl

`prod`

Anwendung

Das Produkt der einzelnen Elemente

Beschreibung

`prod(X)` berechnet das Produkt der Elemente des Vektors X. Handelt es sich bei X um eine Matrix, so wird für jeden Kolonnenvektor das Produkt seiner Elemente berechnet und die Resultate einem Zeilenvektor zugewiesen.

`prod(X,dim)` multipliziert entlang der Dimension dim, d.h. für `prod(X,1)` wird das Produkt kolonnenweise gebildet und für `prod(X,2)` zeilenweise.

Beispiel

```
>> A=[5 1 8;6 9 2;7 0 5]
A =
      5     1     8
      6     9     2
      7     0     5
>> prod(A,1)
ans =
      210     0    80
>> prod(A,2)
ans =
       40
      108
        0
```

Siehe auch

`cumprod`

6.2.2 Finite Differenzen

Befehle	Kurzbeschreibung
`diff`	Differenz
`gradient`	Gradient

Befehl

`diff`

Anwendung

Berechnen der Differenz und Approximation der Ableitung

Beschreibung

`diff(X)` berechnet die Differenzen zwischen benachbarten Elementen von X. Für einen Vektor X mit n Elementen liefert `diff(X)` also einen Vektor mit den Differenzen [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)], der um ein Element kürzer ist als X selbst.

Bei einer n×m-Matrix X berechnet `diff(X)` jeweils die Differenzen der benachbarten Elemente innerhalb des jeweiligen Kolonnenvektors. Das Ergebnis ist die (n-1)×m-Matrix `[X(2:n,:)-X(1:n-1,:)]`

`diff(X,N)` wendet `diff` rekursiv N mal an und bestimmt so die N-te Differenz. `diff(X,2)` ist daher dasselbe wie `diff(diff(X))`.

Mit `diff(X,N,dim)` wird die N-te Differenz entlang der Dimension dim bestimmt. Durch `diff(X,N,1)` wird also kolonnenweise die N-te Differenz gebildet und durch `diff(X,N,2)` zeilenweise.

`diff` kann zur Approximation einer Ableitung benützt werden (siehe Beispiel).

Beispiel

```
>> h=0.001;
>> X=0:h:pi;
>> Y=sin(X);
>> % Approximation von cos(x)=d/dx(sin(x))
>> cos_approx=diff(Y)/h;
>> A=[5 1 8;6 9 2;7 0 5];
>> diff(A,1)
ans =
     1     8    -6
     1    -9     3
>> diff(A,2)
ans =
     0   -17     9
```

Siehe auch

`sum`, `prod`

Befehl

`gradient`

Anwendung

Partielle numerische Differentiation

Beschreibung

Der Gradient einer Funktion f(x,y) ist definiert als

$$\nabla f = \frac{\partial f}{\partial x}\hat{i} + \frac{\partial f}{\partial y}\hat{j} \tag{21}$$

Für eine Matrix f können mit `[fx,fy]=gradient(f)` die x- und die y-Komponente des numerischen Gradienten berechnet werden. fx entspricht dabei der partiellen Ableitung $\partial f / \partial x$, und fy entspricht $\partial f / \partial y$. Dabei wird angenommen, dass der Abstand zwischen zwei x-Werten bzw. zwischen zwei y-Werten jeweils 1 ist.

Ist f ein Vektor, so entspricht `df=gradient(f)` dem eindimensionalen Gradienten.

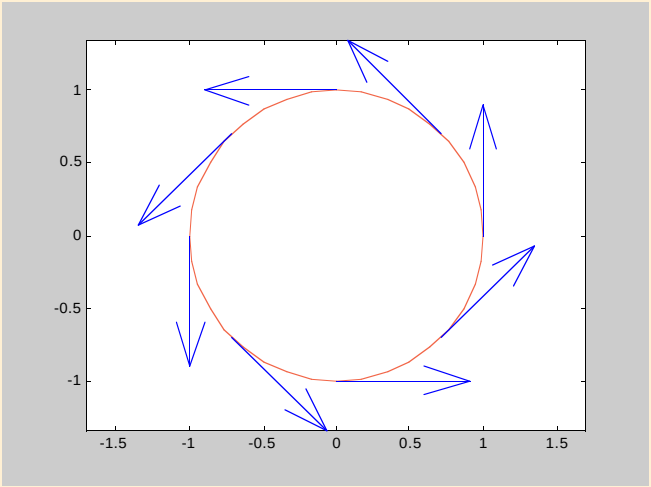
`[fx,fy]=gradient(f,h)` trifft die Annahme, dass die Abstände der Punkte in x- und in y-Richtung jeweils h sind.

Mit `[fx,fy]=gradient(f,hx,hy)` wird ein Abstand hx in x-Richtung und ein Abstand hy in y-Richtung verwendet, wenn hx und hy Skalare sind. Handelt es sich bei diesen Grössen um Vektoren, so werden sie als die Koordinaten der einzelnen Punkte angesehen.

3-D Gradienten werden analog berechnet.

Beispiel

```
>> t=-pi/4:pi/4:2*pi;
>> x=cos(t);
>> y=sin(t);
>> l=length(t)-1;
>> x1=cos(0:pi/18:2*pi);
>> y1=sin(0:pi/18:2*pi);
>> plot(x1,y1,'r--')
>> axis equal
>> hold on
>> fx=gradient(x);
>> fy=gradient(y);
>> quiver(x(2:l),y(2:l),fx(2:l),fy(2:l))
>> hold off
```



Siehe auch

`del2`

6.2.3 Korrelation

Befehle	Kurzbeschreibung
`cov`	Kovarianz
`corrcoef`	Korrelationskoeffizient

Befehl

`cov`

Anwendung

Die Varianz eines Vektors bzw. die Kovarianz einer Matrix berechnen

Beschreibung

Ist X ein Vektor, so berechnet `cov(X)` die Varianz von X. Für die Varianz von X gilt:

$$s_x^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right) \qquad (22)$$

Die Standardabweichung s_x ist die Quadratwurzel aus der Varianz und beschreibt die mittlere quadratische Abweichung bzw. die Streuung um den Mittelwert.

Ist X eine Matrix, bei der jede Kolonne für eine unabhängige Variable steht und jede Zeile für die von diesen Variablen abhängige Beobachtung, so berechnet `cov(X)` die Kovarianzmatrix von X. Sie beschreibt die Stärke der linearen Beziehung zwischen den unabhängigen Variablen. Die Kovarianzmatrix ist symmetrisch und enthält die Varianz der einzelnen Variablen in der Diagonalen. Mit `diag(cov(X))` können daher die Varianzen der einzelnen Variablen bestimmt werden und mit `sqrt(diag(cov(X)))` die Standardabweichungen.

`cov(X,Y)` bestimmt die Stärke der linearen Beziehung zwischen den unabhängigen Vektoren X und Y. Diese müssen dieselbe Anzahl von Elementen besitzen.

Normalerweise normiert `cov` mit n-1. `cov(X,1)` und `cov(X,Y,1)` hingegen normieren mit n.

`cov` zieht von jeder Kolonne ihren Mittelwert ab, bevor das Ergebnis berechnet wird.

Beispiel

```
>> x=[1 6 3 8 0 2];
>> y=[2 7 3 6 4 1];
>> cov(x) % Die Varianz von x
ans =
      9.4667
>> cov(y) % Die Varianz von y
ans =
      5.3667
>> % Die Varianz von x und y
>> % und die Kovarianz zwischen x und y
>> cov(x,y)
ans =
    9.4667    5.2667
    5.2667    5.3667
>> C=[x' y']
C =
      1     2
      6     7
      3     3
      8     6
      0     4
      2     1
>> cov(C)
ans =
      9.4667    5.2667
      5.2667    5.3667
```

Siehe auch

`std`, `mean`

Befehl

`corrcoef`

Anwendung

Berechnen des Korrelationskoeffizienten

Beschreibung

Der Korrelationskoeffizient r_{xy} ist ein normiertes Mass für die Stärke des linearen Zusammenhangs zwischen zwei Zufallsvariablen X und Y. Voneinander unabhängige Variablen besitzen einen Korrelationskoeffizienten $r_{xy} = 0$. Sind die Zufallsvariablen X und Y linear voneinander abhängig, so ist $r_{xy} = \pm 1$. Alle Punkte (X,Y) liegen auf einer Geraden mit positiver (für $r_{xy} = 1$) bzw. negativer (für $r_{xy} = -1$) Steigung.

Für den Korrelationskoeffizienten gilt:

$$r_{xy} = s_{xy}/(s_x s_y) \tag{23}$$

s_x und s_y sind die Varianzen der Zufallsvektoren X und Y,

s_{xy} ist die Kovarianz der Zufallsvektoren X und Y:

$$s_{xy} = \frac{1}{n-1}\sum\left(x_i - \frac{1}{n}\sum x_i\right)\left(y_i - \frac{1}{n}\sum y_i\right) \tag{24}$$

`corrcoef(X,Y)` berechnet für die Kolonnenvektoren X und Y die Korrelationskoeffizienten.

Beispiel

```
>> x=[1 6 3 8 0 2];y=[2 7 3 6 4 1];
>> corrcoef(x,y)
ans =
      1.0000    0.7389
      0.7389    1.0000
>> x=[1 2 3];y=[3 6 9];
>> corrcoef(x,y)
ans =
      1     1
      1     1
```

Siehe auch

`std`

6.3 Fourier-Transformation

Befehle	Kurzbeschreibung
`fft`	Diskrete Fourier Transformation

Befehl

`fft`

Anwendung

Berechnung der eindimensionalen diskreten Fourier-Transformation.

Beschreibung

Die Fourier-Transformation zerlegt ein Signal in eine unendliche Summe von Sinusfunktionen unterschiedlicher Frequenzen. Im Gegensatz zu Fourier-Reihen, die nur von periodischen Signalen gebildet werden können, kann von (fast) jedem beliebigen Signal die Fourier-Transformierte berechnet werden. Für diskrete Signale wird eine diskrete Fourier-Transformation verwendet; die schnellsten Algorithmen dafür sind die FFT-Algorithmen (Fast Fourier Transform).

Für einen Vektor x der Länge N liefert `fft(x)` die diskrete Fourier-Transformierte X (ebenfalls mit der Länge N):

$$X(k) = \sum_{n=1}^{N} x(n) e^{-j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1 \le k \le N \qquad (25)$$

Dabei kommt ein besonders schneller Algorithmus zur Anwendung, wenn N eine Potenz von 2 ist.

Handelt es sich bei x um ein mit einer Periode T abgetastetes Signal, so können die zugehörigen Frequenzen f (in Hz) wie folgt berechnet werden:

$$f(k) = \frac{k-1}{NT} \qquad 1 \le k \le N \qquad (26)$$

Da x ein abgetastetes Signal ist, wiederholen sich die Werte im Frequenzbereich. Aus diesem Grund beinhalten nur die Wert für Frequenzen f(k) < 1/(2T) relevante Informationen, die restlichen Werte sind redundant.

Die Fourier-Rücktransformation `ifft(X)` erfolgt nach:

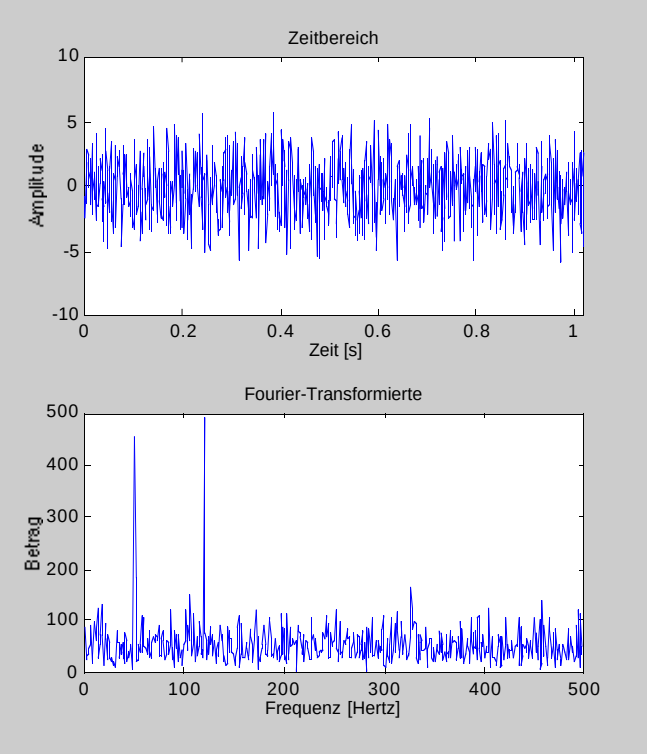
$$x(n) = \frac{1}{N} \sum_{k=1}^{N} X(k) e^{j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1 \le n \le N \qquad (27)$$

`fft(x,N)` entspricht einer Fourier-Transformation mit N Punkten. Besitzt der Vektor x weniger als N Werte, so wird eine entsprechende Anzahl von Nullen an X angehängt. Hat x mehr als N Werte, so werden nur die ersten N verwendet.

Bei einer Matrix `x` wird für jede Kolonne einzeln die Fourier-Transformierte berechnet.

Beispiel

```
>> % Abtastperiode 1/1000 s; 2^10=1024 Werte
>> t = 0:0.001:(1024-1)*0.001;
>> % Ein verrauschtes Signal
>> x = sin(2*pi*50*t)+sin(2*pi*120*t);
>> xr = x + 2*randn(size(t));
>> y=fft(xr);m=abs(y);    % Betrag
>> f=(0:1024-1)'*1000/1024; % Frequenzen
>> subplot(2,1,1)
>> plot(t,xr); axis([0 max(t) -10 10])
>> title('Zeitbereich')
>> ylabel('Amplitude');xlabel('Zeit [s]')
>> subplot(2,1,2); plot(f(1:512),m(1:512))
>> title('Fourier-Transformierte')
>> ylabel('Betrag')
>> xlabel('Frequenz [Hertz]')
```



Siehe auch

`ifft`, `fft2`, `ifft2`, `fftshift`, `abs`, `angle`, `unwrap`

7 Interpolation und Polynome

Die Interpolation ist ein wichtiges Hilfsmittel, um Werte zu berechnen, die zwischen bekannten Daten liegen, oder um eine Kurve bzw. eine Fläche in eine Messdatenreihe einzubetten. MATLAB kennt dafür mehrere Interpolationsmethoden.

Polynome werden in MATLAB durch ihre Koeffizienten charakterisiert, die in Zeilenvektoren eingegeben werden. Polynome lassen sich addieren, subtrahieren, multiplizieren und dividieren. Daneben können die Nullstellen, die Ableitungen und die Partialbruchzerlegungen von Polynomen berechnet werden.

7.1 Interpolation

Befehle	Kurzbeschreibung
`interp1`	1-D Interpolation
`interp2`	2-D Interpolation
`griddata`	Interpolation mit gleichmässig verteiltem Gitter
`spline`	Kubische Interpolation

Befehl

`interp1`

Anwendung

Daten eindimensional interpolieren

Beschreibung

`yi=interp1(x,y,xi)` interpoliert anhand der Funktion y(x) für die Punkte aus dem Vektor xi die entsprechenden Werte des Vektors yi. y ist in Funktion von x, d.h. jedem Element von x ist eindeutig ein Element von y zugeordnet.

Handelt es sich bei y um eine Matrix, so wird für jede Kolonne einzeln interpoliert.

Für `yi=interp1(y,xi)` erzeugt MATLAB selber einen Vektor x. Er hat dieselbe Länge wie y und entspricht dem MATLAB Befehl `x=1:length(y)`.

`yi=interp1(x,y,xi,'method')` definiert zusätzlich die Interpolationsmethode.

Folgende Interpolationsarten sind möglich:

'`nearest`' - rundet den Wert des Interpolationspunktes auf die nächstliegende ganze Zahl (siehe -> `round`)

'`linear`' - lineare Interpolation

'`spline`' - verwendet eine Vielzahl von Funktionen für die Interpolation zwischen zwei Punkten

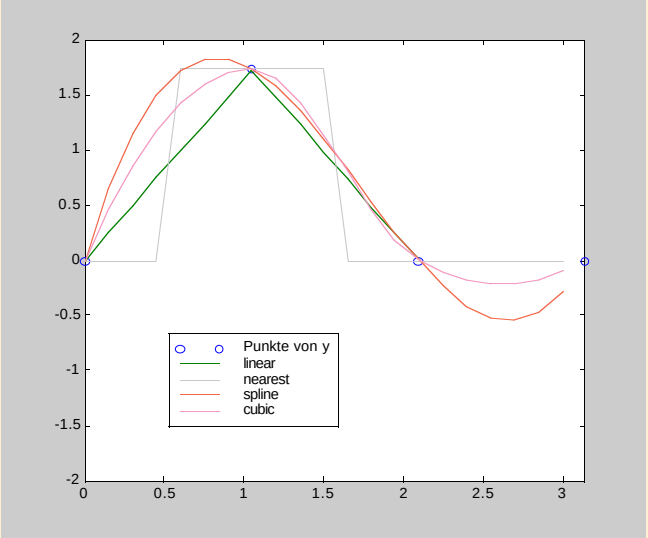
'`cubic`' - kubische Interpolation

Die default-Einstellung ist `'linear'`.

All diese Methoden verlangen, dass die Schrittlänge zwischen den einzelnen Elementen von x konstant bleibt.

Beispiel

```
>> x=0:pi/3:pi;
>> y=sin(x)+sin(2*x);
>> xi=0:0.15:pi;
>> yi=interp1(x,y,xi);
>> plot(x,y,'o',xi,yi), hold on
>> yi=interp1(x,y,xi,'nearest');
>> plot(xi,yi,'k:')
>> yi=interp1(x,y,xi,'spline');
>> plot(xi,yi,'r--')
>> yi=interp1(x,y,xi,'cubic');
>> plot(xi,yi,'m-.')
>> axis([0 pi -2 2])
>> legend('Punkte von y','linear',...
   'nearest','spline','cubic')
>> hold off
```



Siehe auch

`interpft`, `interp3`, `interpn`

Befehl

`interp2`

Anwendung

Daten zweidimensional interpolieren

Beschreibung

`Zi=interp2(X,Y,Z,Xi,Yi)` interpoliert anhand der Funktion Z(X,Y) für die Punkte aus der Matrizen Xi und Yi die entsprechenden Werte der Matrix Zi. Die Matrix Z ist in Funktion von X und Y, d.h. jedem Punkt von X und Y ist eindeutig ein Element von Z zugeordnet.

Handelt es sich bei Xi um einen Zeilenvektor, so erzeugt MATLAB eine Matrix mit konstanten Kolonnen der Länge Yi. Ist Yi eine Kolonnenvektor, dann bildet MATLAB eine Matrix mit konstanten Zeilen der Länge Xi.

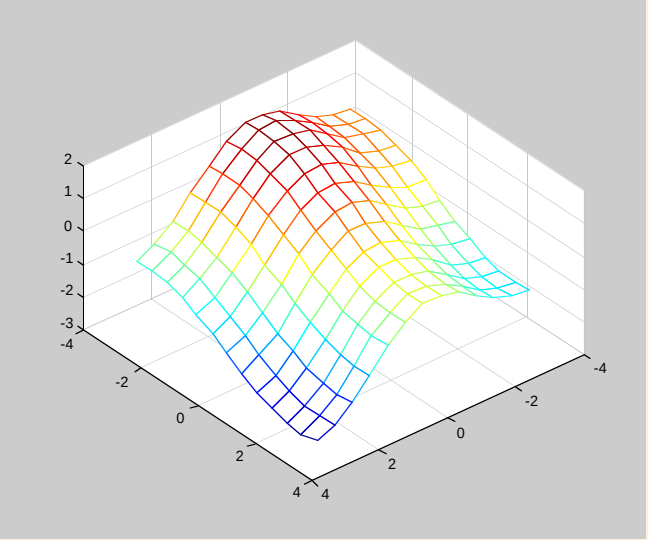
Für `Zi=interp2(Z,Xi,Yi)` generiert MATLAB für die Funktion Z(X,Y) Element von $R^{n \times m}$ die Vektoren X und Y selber. X hat die Länge m und Y die Länge n.

`Zi=interp2(Z,ntimes)` bestimmt die Anzahl Interpolationen, die zwischen zwei Elementen von Z gemacht werden. `interp2(Z,1)` ist das Gleiche wie `interp2(Z)`.

`interp2(...,'method')` verwendet bis auf die Methode `'spline'` dieselben Interpolationsmethoden wie `interp1`. Auch hier gilt, dass die Schrittlängen zwischen den einzelnen Elementen von X und Y konstant bleiben.

Beispiel

```
>> x=-pi:pi/3:pi; y=-pi:pi/3:pi;
>> [X,Y]=meshgrid(x,y);
>> Z=cos(X).*cos(0.1.*Y)-sin(0.3.*X)-sin(0.6.*Y);
>> xi=-pi:pi/6:pi; yi=-pi:pi/6:pi;
>> [Xi,Yi]=meshgrid(xi,yi);
>> Zi=interp2(X,Y,Z,Xi,Yi,'cubic');
>> mesh(Xi,Yi,Zi)
>> view(140,40)
```



Siehe auch

`interp3`, `interpn`, `meshgrid`

Befehl

`griddata`

Anwendung

Aus nicht gleichmässig verteilten Punkten eine Oberfläche interpolieren

Beschreibung

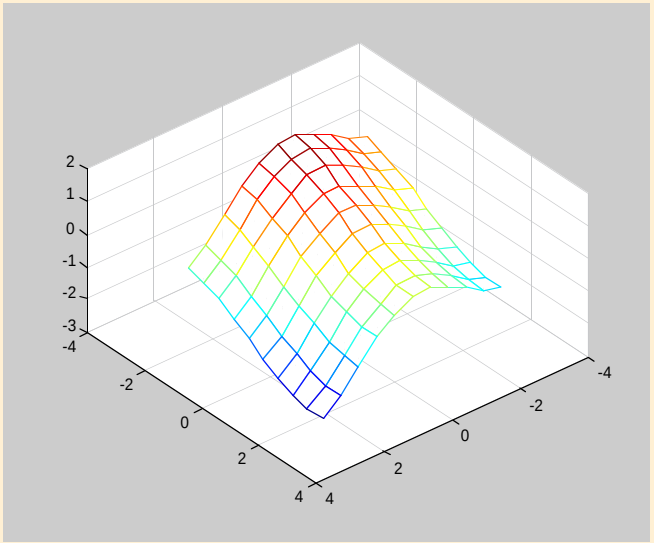
`Zi=griddata(X,Y,Z,Xi,Yi)` passt eine Oberfläche der Form Z = F(X,Y) in die Punkte der nicht normalverteilten Vektoren X,Y und Z ein. Dazu interpoliert `griddata` anhand der Funktion Z(X,Y) für die Punkte aus den Matrizen Xi und Yi die entsprechenden Werte der Matrix Zi. Die Oberfläche geht jedoch stets durch die Punkte von X,Y und Z. Xi und Yi sind Matrizen. Sie bilden ein gleichmässig unterteiltes Gitter (siehe -> `meshgrid`).

Handelt es sich bei Xi um einen Zeilenvektor, so erzeugt MATLAB eine Matrix mit konstanten Kolonnen der Länge Yi. Ist Yi ein Kolonnenvektor, dann bildet MATLAB eine Matrix mit konstanten Zeilen der Länge Xi.

`[...]=griddata(...,'method')` verwendet dieselben Interpolationsmethoden wie `interp2`. Zusätzlich kann die Methode '`invdist`' gewählt werden. Sie glättet die Oberfläche.

Beispiel

```
>> x=[-3 -2 -1.5 -0.5 0 1 2 2.5 3];
>> y=[-2.5 -2 -1.5 -1 0 0.5 1 1.5 2 3];
>> [X,Y]=meshgrid(x,y);
>> Z=cos(X).*cos(0.1.*Y)-sin(0.3.*X)-sin(0.6.*Y);
>> xi=-pi:pi/6:pi;
>> yi=-pi:pi/6:pi;
>> [Xi,Yi]=meshgrid(xi,yi);
>> Zi=griddata(X,Y,Z,Xi,Yi,'cubic');
>> mesh(Xi,Yi,Zi)
>> view(140,50)
```



Siehe auch

`meshgrid`

Befehl

`spline`

Anwendung

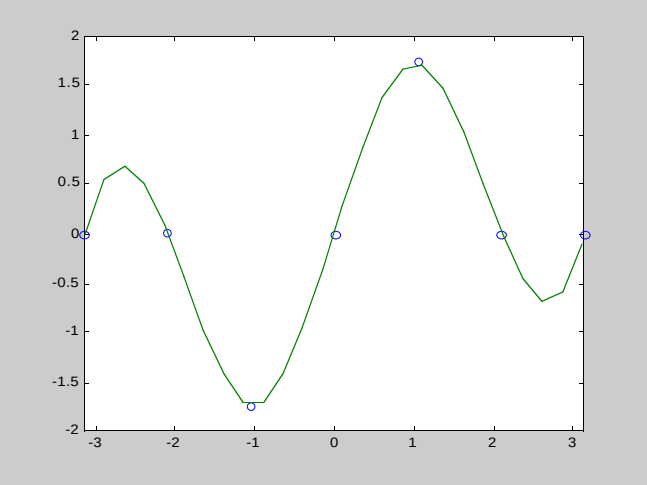
Kubische Spline Interpolation

Beschreibung

`yi=spline(x,y,xi)` rechnet mit der kubischen Spline Interpolation. Anhand der Funktion y(x) interpoliert spline für die Punkte aus dem Vektor xi die entsprechenden Werte des Vektors yi. Die kubische Spline Interpolation verwendet eine Vielzahl von Funktionen für die Interpolation zwischen zwei Punkten. Von Punkt zu Punkt werden die Funktionen neu ermittelt und können somit ändern.

Beispiel

```
>> x=-pi:pi/3:pi;
>> y=sin(x)+sin(2*x);
>> xi=-pi:0.25:pi;
>> yi=spline(x,y,xi);
>> plot(x,y,'o',xi,yi)
>> axis([-pi pi -2 2])
```



Siehe auch

`ppval`

7.2 Polynome

Befehle	Kurzbeschreibung
`roots`	Nullstellen eines Polynoms
`poly`	Polynom aus einer Matrix oder den Nullstellen
`polyval`	Polynom für spezifische Punkte evaluieren
`residue`	Partialbruchzerlegung
`polyfit`	Polynom in (Mess-) Daten einpassen
`polyder`	Polynom ableiten
`conv`	Polynome multiplizieren
`deconv`	Polynome dividieren

Befehl

`roots`

Anwendung

Die Nullstellen einer Polynomfunktion berechnen

Beschreibung

In MATLAB wird ein Polynom f = f(x) mit seinen Koeffizienten c(i) definiert. Der Vektor C=[c(1) c(2)...c(n) c(n+1)] entspricht dem Polynom:

$$f(x) = c_1 x^n + c_2 x^{n-1} + c_3 x^{n-2} + \ldots + c_n x + c_{n+1} \qquad (28)$$

`roots(C)` berechnet die Nullstellen des Polynoms, dessen Koeffizienten im Vektor C gegeben sind.

Beispiel

```
>> C=[1 -2.5 0 2]; % f(x)=x^3-2.5*x^2+2
>> roots(C)
ans =
        2.0000
        1.2808
       -0.7808
```

Siehe auch

`fzero`

Befehl

`poly`

Anwendung

Aus einer Matrix die entsprechenden Koeffizienten des charakteristischen Polynoms ermitteln.

Aus den Nullstellen das entsprechende Polynom berechnen

Beschreibung

`poly(A)` berechnet aus der n×n-Matrix A die Koeffizienten des charakteristischen Polynoms. Die Definition des charakteristischen Polynoms lautet

$$\det(\lambda I - A) = \lambda^n + a_{n-1}\lambda^{n-1} + \ldots + a_1\lambda + a_0 = 0. \qquad (29)$$

Die Eigenwerte von A (-> `eig(A)`) entsprechen den Nullstellen des charakteristischen Polynoms (-> `roots(poly(A))`).

poly(V) ermittelt für Nullstellen aus dem Vektor V die Koeffizienten des charakteristischen Polynoms.

Beispiel

```
>> A=[1 3;8 3];
>> poly(A)
ans =
     1    -4   -21
>> eig(A)
ans =
     -3
      7
>> roots(poly(A))
ans =
      7
     -3
>> % Die Nullstellen aus dem roots-Beispiel
>> B=[2 1.2808 -0.7808];
>> poly(B)
ans =
     1.0000   -2.5000   -0.0000    2.0001
```


Befehl

`polyval`

Anwendung

Das Polynom für bestimmte Punkte berechnen

Beschreibung

`y=polyval(C,x)` berechnet an der Stelle x den Betrag des Polynoms f = f(x). C ist ein Zeilenvektor mit den Koeffizienten $c_1, c_2, ..., c_{n+1}$ des charakteristischen Polynoms

$$f(x) = c_1x^n + c_2x^{n-1} + c_3x^{n-2} + \ldots + c_nx + c_{n+1} \qquad (30)$$

x ist ein Skalar, ein Vektor oder eine Matrix.

Beispiel

```
>> f=[1 -5 2 -9 3]; % x^4-5*x^3+2*x^2-9*x+3
>> polyval(f,2) % f(2)=?
ans =
      -31
>> x=0:5;
>> polyval(f,x)
ans =
      3    -8   -31   -60   -65     8
>> X=[1 2 4;1 3 9;1 4 16];
>> polyval(f,X)
ans =
      -8         -31         -65
      -8         -60        3000
      -8         -65       45427
```

Siehe auch

`polyvalm`

Befehl

`residue`

Anwendung

Partialbruchzerlegung

Beschreibung

Vorausgesetzt wird eine echt gebrochene rationale Funktion $r(x) = B_m(x)/A_n(x)$, wobei B_m und A_n Polynome m-ten und n-ten Grades mit m<n sind:

$$r(x) = \frac{B_m(x)}{A_n(x)} = \frac{b_1 x^m + b_2 x^{m-1} + \ldots + b_m x + b_{m+1}}{a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}} \quad (31)$$

`[r,p,k]=residue(B,A)` ermittelt für das Zählerpolynom B und das Nennerpolynom A die Zähler r, die Nenner (Pole) p und den direkten Term k der Partialbruchzerlegung:

$$\frac{B(x)}{A(x)} = \frac{r_1}{x-p_1} + \frac{r_2}{x-p_2} + \ldots + \frac{r_n}{x-p_n} + k(x) \quad (32)$$

B und A sind Zeilenvektoren mit den Koeffizienten der charakteristischen Zähler- und Nennerpolynome.

Die Anzahl Pole p ist gleich n.

`[B,A]=residue(r,p,k)` macht die Partialbruchzerlegung rückgängig.

Falls die Pole des Nennerpolynoms nahe beieinanderliegen, können bei der Partialbruchzerlegung willkürlich grosse Veränderungen im Zähler r und Nenner p auftauchen.

Beispiel

```
>> B=[1 23];A=[1 6 -7];
>> [r,p,k]=residue(B,A)
r =
      -2
       3
p =
      -7
       1
k =
       []
```


Befehl

`polyfit`

Anwendung

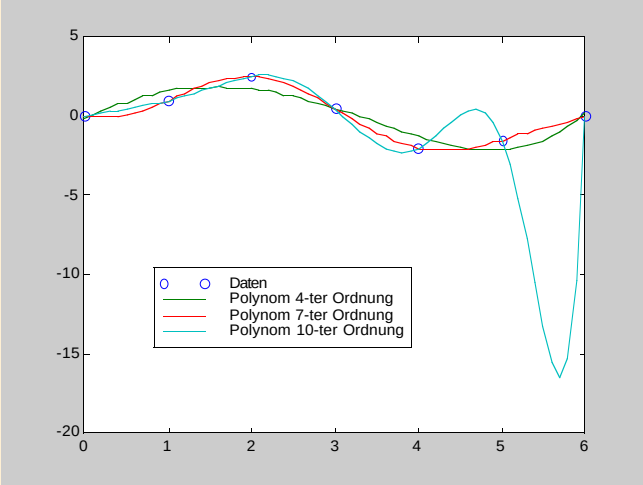
Aus Punkten ein Polynom von beliebiger Ordnung eruieren

Beschreibung

`C=polyfit(x,y,N)` berechnet die Koeffizienten jenes charakteristischen Polynoms N-ter Ordnung, welches in die Daten aus den Vektoren x und y eingebettet wird.

Beispiel

```
>> x=[0 1 2 3 4 5 6];
>> y=[0 1 2.5 0.5 -2 -1.5 0];
>> C1=polyfit(x,y,4);
>> C2=polyfit(x,y,7);
>> C3=polyfit(x,y,10);
>> t=0:0.1:6;
>> f1=polyval(C1,t);
>> f2=polyval(C2,t);
>> f3=polyval(C3,t);
>> plot(x,y,'o',t,f1,t,f2,t,f3)
>> legend('Daten','Polynom 4-ter Ordnung',...
   'Polynom 7-ter Ordnung',...
   'Polynom 10-ter Ordnung')
```



Befehl

`polyder`

Anwendung

Die Ableitung eines Polynoms n-ter Ordnung ermitteln

Beschreibung

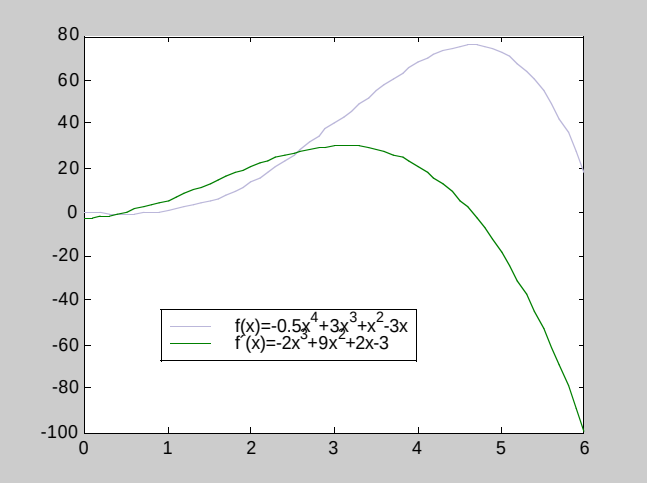
`polyder(C)` berechnet die Ableitung eines Polynoms mit den Koeffizienten im Vektor C=[c(1) c(2) c(3) ... c(n) c(n+1)].

`polyder(A,B)` berechnet die Ableitung des Produkts A*B.

`[N,D]=polyder(B,A)` ermittelt die Ableitung der Division B/A. Das Resultat ist in der Form N/D. A, B, D und N sind Vektoren mit den Koeffizienten der einzelnen Polynome.

Beispiel

```
>> t=0:0.1:6;
>> C=[-0.5 3 1 -3 0];
>> fC=polyval(C,t);
>> D=polyder(C)
D =
      -2     9     2    -3
>> fD=polyval(D,t);
>> plot(t,fC,':',t,fD)
>> legend('f(x)=-0.5x^4+3x^3+x^2-3x',...
   'f´(x)=-2x^3+9x^2+2x-3')
```



Befehl

`conv`

Anwendung

Multiplikation von Polynomen

Beschreibung

`C=conv(A,B)` multipliziert Polynom A mit Polynom B. A und B sind Vektoren mit den Koeffizienten der einzelnen Polynome.

Beispiel

```
>> a=5; % Skalar
>> A=[1 -4 4]; % x^2 - 4*x +4
>> B=[1 3 -5 7]; % x^3 + 3*x^2 - 5*x +7
>> CaA=conv(a,A)
CaA =
      5   -20    20
>> CAB=conv(A,B)
CAB =
      1    -1   -13    39   -48    28
```

Siehe auch

`xcorr`, `conv2`

Befehl

deconv

Anwendung

Division von Polynomen

Beschreibung

`[Q,R]=deconv(A,B)` dividiert Polynom A durch Polynom B. A und B sind Vektoren mit den Koeffizienten der einzelnen Polynome. Der Quotient ist in der Variable Q abgespeichert. Der nicht dividierbare Rest steht im Vektor R.

Beispiel

```
>> a=2;
>> A1=[7 18 -44 15];
>> A2=[2 -9 4 0 -3];
>> B=[1 3 -5];
>> [q,r]=deconv(A1,a)
q =
      3.5000    9.0000  -22.0000    7.5000
r =
      0     0     0     0
>> [q,r]=deconv(A1,B)
q =
      7    -3
r =
      0     0     0     0
>> [q,r]=deconv(A2,B)
q =
      2   -15    59
r =
      0     0     0  -252   292
>> A = conv(B,q)+r
A =
      2    -9     4     0    -3 % A=A2
```